A reliable Turing machine

Ilir Çapuni
University of Montenegro
ilir@bu.edu

Peter Gács
Boston University
gacs@bu.edu

May 26, 2020

Abstract

We consider computations of a Turing machine subjected to noise. In every step, the action (the new state and the new content of the observed cell, the direction of the head movement) can differ from that prescribed by the transition function with a small probability (independently of previous such events). We construct a universal Turing machine that for a low enough (constant) noise probability performs arbitrarily large computations. For this unavoidably, the input needs to be encoded—by a simple code depending on its size. The work uses a technique familiar from reliable cellular automata, complemented by some new ideas.

1 Introduction

This work addresses a question from the area of "reliable computation with unreliable components". A certain class of machines is chosen, (like a Boolean circuit, cellular automaton, Turing machine). It is specified what kind of faults (local in space and time) are allowed, and a machine of the given kind is built that—paying some price in performance—carries out essentially the same task as any given machine of the same kind without any faults would.

We confine attention to *transient*, *probabilistic* faults: the fault occurs at a given time but no component is damaged permanently, and faults occur independently of each other, with a bound on their probability. This in contrast to bounding the *number* of faults, allowing them to be set by an adversary.

Historically the first result of this kind is [10], which for each Boolean circuit C of size n constructs a new circuit C' of size $O(n \log n)$ that performs the same task as C with a (constant) high probability, even though each gate of C' is allowed to fail with some (constant) small probability.

Cellular automata as a model have several theoretical advantages over Boolean circuits, and results concerning reliable computation with them have interest also from a purely mathematical or physical point of view (non-ergodicity). The simple construction in [7] gives, for any 1-dimensional cellular automaton A a 3-dimensional cellular automaton A' that performs the same task as A with high probability, even though each cell of A' is allowed to fail in each time step with some constant small probability. (A drawback of this construction is the requirement of synchronization: all cells of A' must update simultaneously.) Reliable cellular automata in less than 3 dimensions can also be constructed (even without the synchrony requirement), but so far only at a steep increase in complexity (both of the construction and the proof). The first such result was [5], relying on some ideas proposed in [8].

Here, the reliability question will be considered for a *serial* computation model—Turing machine—as opposed to parallel ones like Boolean circuits or cellular automata. There is a single elementary processing unit (the *active* unit) interacting with a memory of unlimited size. The error model needs to be relaxed. Allowing each memory component to fail in each time step with constant probability makes, in the absence of parallelism, reliable computation seems impossible. Indeed, while in every step some constant fraction of the memory gets corrupted, the active unit can only correct a constant *number* of them per step.

In the relaxed model considered here, faults can affect only the operation of the active unit. The challenge for Turing machines is still significant, since even if only with small probability, occasionally a group of faults can put the head into the middle of a large segment of the tape rewritten in an arbitrarily "malicious" way. A method must be found to recover from all these situations.

Here we define a Turing machine that is reliable—under this fault model— in the same sense as the other models above. The construction and proof are similar in complexity to the ones for 1-dimensional cellular automata; however, we did not find a reduction to these earlier results. A natural idea is to let the Turing machine simulate a 1-dimensional cellular automaton, by having

the head make large sweeps, and updating the tape as the simulated cellular automaton would. But apart from the issue of excessive delay, we did not find any simple way to guarantee the large sweeps in the presence of faults (where "simple" means not building some new hierarchy), even for some price paid in efficiency. So here we proceed "from scratch".

Many ideas used here are taken from [5] and [6], but hopefully in a somewhat simpler and more intuitive conceptual framework. Like [5] it confines all probability reasoning to a single lemma, deals on each level only with a numerical restriction on faults (of that level). On the other hand, like [6] it defines a series of generalized objects (generalized Turing machines here rather than generalized cellular automata), each one simulating the next in the series. In [6] a "trajectory" (central for defining the notion of simulation) was a random history whose distribution satisfies certain constraints (most of which are combinatorial). Here it is a single history satisfying only some combinatorial constraints.

The work [1] seems related by its title but is actually on another topic. The work [4] applies the self-simulation and hierarchical robustness technique developed for cellular automata in an interesting, but simpler setting. Several attempts at the chemical or biological implementation of universal computation have to deal with the issue of error-correction right away. In these cases generally the first issue is the faults occurring at the active site (the head). See [2, 9].

Our result can use any standard definition of 1-tape Turing machines whose tape alphabet contains some fixed "input-output alphabet" Σ; we will introduce one formally in Section 2.6. We will generally view a tape symbol as a tuple consisting of several *fields*. The notation

$$a.Output, a.Info$$

shows *Output* and *Info* as fields of tape cell state a. Combining the same field of all tape cells, we can talk about a *track* (say the *Output* track and *Info* track). For ease of spelling out a result, we consider only computations whose outcome is a single symbol, written into the *Output* field of tape position 1. It normally holds a special value—say $*$ —meaning *undefined*.

Block codes (as defined in Section 3.2 below) are specified by a pair (ψ_*, ψ^*) of encoding and decoding functions. In the theorem below, the input of the computation, of some length n, is broken up into blocks that are encoded by a block code that depends in some simple way on n. Its redundancy depends on the size of the input as a log power. The main result in the theorem below shows a Turing machine simulating a fault-free Turing machine computation in a fault-tolerant way. It is best to think of the simulated machine G as some universal Turing machine.

Theorem 1 *For any Turing machine G there are constants $\alpha_1, \alpha_2 > 0$, for each n a block code (φ_*, φ^*) of block size $O((\log n)^{\alpha_1})$, a fault bound $0 \leq \varepsilon < 1$ and a Turing machine M_1 with a function $a \mapsto a.Output$ defined on its alphabet, such that the following holds.*

Let M_1 start its work from the initial tape configuration $\varphi_*(x)$ with the head in position 0. Assume further that during its operation, faults occur independently at random with probabilities $\leq \varepsilon$. Suppose that at time t the machine G writes a value $y \neq *$ into the Output field of the cell at position 0. Then at any

time greater than $t(\log t)^{\alpha_2}$*, the tape symbol a of machine* M_1 *at position 0 will have* $a.Output = y$ *with probability at least* $1 - O(\varepsilon)$.

2 Overview

The overview, but even the main text, is not separated completely into two parts: the definition of the Turing machine, followed by the proof of its reliability. The definition of the machine is certainly translatable (with a lot of tedious work) into just a Turing machine transition table (or *program*), but its complexity requires first to develop a *conceptual apparatus* behind it which is also used in the proof of reliability. We will try to indicate below at the beginning of each section whether it is devoted more to the program or more to the conceptual apparatus.

2.1 Isolated bursts of faults

(Introducing some basic elements of the *program*.) In [3] we defined a Turing machine M_1 that simulates "reliably" any other Turing machine even when it is subjected to isolated *bursts* of faults (that is a group of faults occurring in consecutive time steps) of constant size. We will use some of the ideas of [3], without relying directly on any of its details, and will add several new ideas. Let us give a brief overview of this machine M_1.

Each tape cell of the simulated machine M_2 will be represented by a block of some size Q called a *colony*, of the simulating machine M_1. Each step of M_2 will be simulated by a computation of M_1 called a *work period*. During this time, the head of M_1 makes a number of sweeps over the current colony-pair, decodes the represented cell symbols, then computes and encodes the new symbols, and finally moves the head to the new position of the head of M_2. The major processing steps will be carried out on a working track three times within one work period, recording the result onto separate tracks. The information track is changed only in a final majority vote.

The organization is controlled by a few key fields, for example a field called *Addr* showing the position of each cell in the colony, and a field *Sweep*, the number of the last sweep of the computation (along with its direction) that has been performed already. The most technical part is to protect this control information from faults. For example, a burst of faults of constant size can reverse the head in the middle of a sweep. To discover such structural disruptions locally before the head would go far in the wrong direction, it will make frequent short zigzags. A premature turn-back will be detected this way, triggering the healing procedure.

2.2 Hierarchy

(Starting to develop the *conceptual apparatus*.) In order to build a machine resisting faults occurring independently in each step with some small probability, we take the approach used for one-dimensional cellular automata. We aim at building a *hierarchy of simulations*: machine M_1

simulates machine M_2 which simulates machine M_3, and so on. Machine M_k has alphabet

$$\Sigma_k = \{0,1\}^{s_k}, \tag{2.1}$$

that is its tape cells have "capacity" s_k. All these machines should be implementable on a universal Turing machine with the same program (with an extra input, the number k denoting the level). For ease of analysis, we introduce the notion of *cell size*: level k has its own cell size B_k and block (colony) size Q_k with $B_1 = 1$, $B_{k+1} = B_k Q_k$. This allows locating a tape cell of M_k on the same interval where the cells of M_1 simulate it. One cell of machine M_{k+1} is simulated by a colony of machine M_k; so one cell of M_3 is simulated by $Q_1 Q_2$ cells of M_1. Further, one step of, say, machine M_3 is simulated by one work period of M_2 of, say, $O(Q_2^2)$ steps.

Per construction, machine M_1 can withstand bursts of faults with size $\leq \beta$ for some constant parameter β, separated by at least some constant γ work periods. It would be natural now to expect that machine M_1 can withstand also some *additional*, larger bursts of size $\leq \beta Q_1$ if those are separated by at least γ work periods of M_2. However, a *new obstacle* arises. Damage caused by a big burst of faults spans several colonies. The repair mechanism of machine M_1 outlined in Section 2.1 is too local to recover from such extensive damage, leaving the whole hierarchy endangered. So we add a new mechanism to M_1 that will just try to restore the colony structure of a large enough portion of the tape (of the extent of several colonies). The task of restoring the original information is left to higher levels (whose simulation now can continue).

All machines above M_1 in the hierarchy live only in simulation: the hardware is M_1. Moreover, the M_k with $k > 1$ will not be ordinary Turing machines, but *generalized* ones, with some new features seeming necessary in a simulated Turing machine: allowing for some "disordered" areas of the tape not obeying the transition function, and occasionally positive distance between neighboring tape cells.

A tricky issue is "forced self-simulation". Each machine M_k can be implemented on a universal machine using as inputs the pair (p, k) where p is the common program and k is the level. Eventually, p will just be hard-wired into the definition of M_1, and therefore faults cannot corrupt it. While creating p for machine M_1, we want to make it simulate a machine M_2 that has the same program p. The method to achieve this is not really different from the one applied already in some of the cellular automata and tiling papers cited, and is related to the proof of Kleene's fixed-point theorem (also called the recursion theorem).

Forced self-simulation can give rise to an infinite sequence of simulations, achieving the needed robustness. But the simulation of M_{k+1} by M_k uses only certain tracks of the tape. Another track (which we will call *Rider*) will be set aside for simulating G. If the simulation of G on the *Rider* track of a colony-pair does not finish in a certain number of steps, a built-in mechanism will *lift* its tape content to the *Rider* field of the simulated cell-pair, allowing it to be continued in a colony-pair of the next level (with the corresponding higher reliability).

2.3 Structuring the noise

(Some definitions for analyzing the noise.) The set of faults in the noise model of the theorem is a set of points in time: each point in time belongs to it with probability ε. It turns out more convenient to use an equivalent model: a set of *space-time points Noise* where each space-time point belongs to *Noise* independently with probability ε (see the introduction to Section 2.5 for motivation).

The construction outlined above counts with *bursts* (rectangles of space-time containing *Noise*) increasing in size and decreasing in frequency—which is a combinatorial set of constraints. To derive such constraints from the above probabilistic model the we stratify *Noise* as follows. We will have two series of parameters: $B_1 < B_2 < \cdots$ and $T_1 < T_2 < \cdots$. Here B_k is the size of cells of M_k as represented on the tape of M_1, and T_k is a bound on the time needed to simulate one step of M_k.

Definition 2.1 For some constants $\beta, \gamma > 0$, a *burst* of noise of type (a, b) is a space-time set that is essentially coverable by a $a \times b$. For an integer $k > 0$ it is of *level* k when it is of type $\beta(B_k \times T_k)$. ⌟

It is *isolated* if it is essentially alone in a rectangle of type $\gamma(B_{k+1} \times T_{k+1})$ (precise definition in Section 2.5). First we remove such isolated bursts of level 1, then of level 2 from the remaining set, and so on. It will be shown that with not too fast increasing sequences B_k, T_k, with probability 1, this sequence of operations eventually completely erases *Noise*: thus each fault belongs to a burst of "level" k for some k.

Machine M_k will concentrate only on correcting isolated bursts of level k and on restoring the framework allowing M_{k+1} to do its job. It can ignore the lower-level bursts and will need to work correctly only in the absence of higher-level bursts.

2.4 Difficulties

(Referring both to the program and to the concepts behind it.) We list here some of the main problems that the paper deals with, and some general ways in which they will be solved or avoided. Some more specific problems will be pointed out later, along with their solution.

Non-aligned colonies A large burst of faults in M_1 can modify the order of entire colonies or create new ones with gaps between them. To deal with this problem, machines M_k for $k > 1$ will be *generalized Turing machines*, allowing for non-adjacent cells.

Clean areas The tape of a generalized Turing machine will be divided, based on its content, into some areas called *clean*, the rest *disordered*. Clean areas will be essentially where the analysis can count on an existing underlying simulation, and where therefore the transition function is applicable. Noise can disorder the areas where it occurs.

Extending cleanness The predictability of the machine is decreased when the head enters into disorder. But the model still provides some "magical" properties helping to restore cleanness (in the absence of new noise):

A) escaping from any area in a bounded amount of time;

B) extension of clean intervals as the head passes in and out of them;

C) the cleaning of an interval when passed over a certain number of times.

While an area is cleaned, it will also be re-populated with cells. Their content is not important, what matters is the restoration of predictability.

Rebuilding The need to reproduce the cleaning properties in simulation is the main burden of the construction. The part of the program devoted to this is the rebuilding procedure, invoked when local repair fails. It reorganizes a part of the tape having the size of a few colonies.

2.5 Structuring the noise formally

(This purely mathematical section derives the combinatorial noise constraints from the probabilistic one.) If we modeled noise as a set of time points then a burst of faults would be a time interval of size βT_k and might affect a space interval as large as βT_k, covering many times more simulated cells of level k. Therefore we model noise as a set of space-time points; this does not change the independence assumption of the main theorem.

Definition 2.2 (Centered rectangles, isolation) Let $\mathbf{r} = (r_1, r_2)$, $r_1, r_2 > 0$ be a two-dimensional nonnegative vector. A *rectangle of radius* $\mathbf{r}$ *centered* at point $\mathbf{x}$ is

$$B(\mathbf{x}, \mathbf{r}) = \{\mathbf{y} : |y_i - x_i| < r_i, i = 1, 2\}. \tag{2.2}$$

Let $E \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ be a space-time set (to be considered our noise set). A point $\mathbf{x}$ of E is $(\mathbf{r}, \mathbf{r}^*)$-*isolated* if $E \cap B(\mathbf{x}, \mathbf{r}^*) \subseteq B(\mathbf{x}, \mathbf{r})$, that is all points of E that are $\mathbf{r}^*$-close to x are also $\mathbf{r}$-close. ⌟

Definition 2.3 Let $\beta \geq 9$ be a parameter, and let

$$1 = B_1 < B_2 < \cdots, \quad 1 = T_1 < T_2 < \cdots$$

be sequences of integers to be fixed later. For a space-time set $E \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$, let $E^{(1)} = E$. For $k > 1$ let $E^{(k+1)}$ be obtained by deleting from $E^{(k)}$ the $(\beta(B_k, T_k), \gamma(B_k, T_{k+1}))$-isolated points. Clearly, for each point $\mathbf{x}$ the event $\mathbf{x} \in E^{(k)}$ depends only on $E \cap B(\mathbf{x}, \gamma(B_{k+1}, T_{k+1}))$. Set E is *k-sparse* if $E^{(k+1)}$ is empty. It is simply *sparse* if $\bigcap_k E^{(k)} = \emptyset$. When $E = E^{(k)}$ and k is known then we will denote $E^{(k+1)}$ simply by E^*. ⌟

The following lemma connects the above defined sparsity notions to the requirement of small fault probability.

Lemma 2.4 (Sparsity) *Let* $Q_k = B_{k+1}/B_k$, $U_k = T_{k+1}/T_k$, *and*

$$\lim_{k \to \infty} \frac{\log(Q_k U_k)}{1.5^k} = 0. \tag{2.3}$$

For sufficiently small ε, for every $k \geq 1$ the following holds. Let $E \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ be a random set with the property that each point belongs to E independently from the other ones with probability $\leq \varepsilon$. Then for each point $\mathbf{x}$ and each k,

$$\mathsf{P}\{ B(\mathbf{x}, (B_k, T_k)) \cap E^{(k)} \neq \emptyset \} < \varepsilon \cdot 2^{-1.5^{k-1}}.$$

As a consequence, the set E is sparse with probability 1.

This lemma allows a doubly exponentially increasing sequence U_k, resulting in relatively few simulation levels as a function of the computation time.

2.6 Generalized Turing machines

(This section, together with Section 2.8, introduces the central concept of the proof.) Let us recall that a one-tape Turing machine is defined by a finite set Γ of *internal states*, a finite alphabet Σ of *tape symbols*, a transition function δ, and possibly some distinguished states and tape symbols. At any time, the head is at some integer position h, and is observing the tape symbol $A(h)$. The meaning of $\delta(a, q) = (a', q', d)$ is that if $A(h) = a$ and the state is q then the $A(h)$ will be rewritten as a' and h will change to $h + d$.

We will use a model that is slightly different, but has clearly the same expressing power. There are no internal states, but the head observes and modifies a *pair* of neighboring tape cells at a time; in fact, we imagine it to be positioned between these two cells called the *current cell-pair*. The *current cell* is the left element of this pair. Thus, a Turing machine is defined as (Σ, τ) where the tape alphabet Σ contains at least the distinguished symbols $\textvisiblespace, 0, 1$ where $\textvisiblespace$ is called the *blank symbol*. The *transition function* is $\tau \colon \Sigma^2 \to \Sigma^2 \times \{-1, 1\}$. A *configuration* is a pair $\xi = (A, h) = (\xi.\mathrm{tape}, \xi.\mathrm{pos})$ where $h \in \mathbb{Z}$ is the *current* (or *observed)* head position, (between cells h and $h + 1$), and $A \in \Sigma^{\mathbb{Z}}$ is the *tape content*, or *tape configuration*: in cell p, the tape contains the symbol $A(p)$. Though the tape alphabet may contain non-binary symbols, we will restrict input and output to binary. The tape is blank at all but finitely many positions.

When the head observes the pair of tape cells with content $\mathbf{a} = (a_0, a_1)$ at positions h, $h + 1$ then denoting $(\mathbf{a}', d) = \tau(\mathbf{a})$ will change tape content at positions h, $h + 1$ to a'_0, a'_1, and move the head to tape position to $h + d$. A *fault* occurs at time t if the output $(\mathbf{a}', d)$ of the transition function at this time is replaced with some other value (which is then used to compute the next configuration).

The machines that occur in simulation will be a generalized version of the above model, allowing non-adjacent cells and areas called "disordered" in which the transition function is non-applicable. A mere convenience feature is two integer parameters: the cell body size $B \geq 1$ and and an upper bound $T \geq 1$ on the transition time. These allow placing all the different Turing machines in a hierarchy of simulations onto the same linear space and the same time line.

Definition 2.5 (Generalized Turing machine) A *generalized Turing machine M* is defined by a tuple

$$(\Sigma, \tau, B, T, \pi), \tag{2.4}$$

where Σ is the *alphabet*, $\tau : \Sigma^2 \times \{0,1\} \to \Sigma^2 \times \{-1,1\}$ is the *transition function*. In $\tau(a,b,\alpha)$ the argument α is 1 if the pair of observed cells is adjacent (no gap between them), and 0 otherwise. The integer π will be the number of passes needed to clean an area (see below). Among the elements of the tape alphabet we distinguish $0, 1, \mathrm{Bad}, \mathrm{Vac}$, where Vac plays the role of a blank symbol. Symbol Bad will mark disordered areas of the tape. ⌟

A formal definition of a configuration of a generalized Turing machine is given in Section 3.4, though it is essentially defined by the tape content A and the head position. A point p is *clean* if $A(p) \neq \mathrm{Bad}$. A set of points is *clean* if it consists of clean points. We say that there is a *cell* at a position $p \in \mathbb{Z}$ if the interval $p + [0, B)$ is clean and $A(p) \neq \mathrm{Vac}$. In this case, we call the interval $p + [0, B)$ the *body* of this cell. Cells must be at distance $\geq B$ from each other, that is their bodies must not intersect. If their bodies are at a distance $< B$ from each other then they are called *neighbors*. They are called *adjacent* if this distance is 0.

A sequence of configurations conceivable as a computation will be called a "history". For standard Turing machines, the histories that obey the transition function then could be called "trajectories". For generalized Turing machines the definition of trajectories is more complex; it allows some limited violations of the transition function, while providing the mechanisms for eliminating disorder. Let $\mathit{Noise} \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ denote the set of space-time points at which faults occur. Section 2.8 below will define a certain subset of possible histories called *trajectories*. In order to motivate their choice, we first introduce the notion of simulation.

2.7 Simulation

(Here we introduce the notion of simulation used in the proof. It relies on the concept of trajectories defined in Section 2.8, but also motivates it.) Until this moment, we used the term "simulation" informally, to denote a correspondence between configurations of two machines which remains preserved during the computation. In the formal definition, this correspondence will essentially be a code $\varphi = (\varphi_*, \varphi^*)$. The *decoding* part of the code is the more important. We want to say that machine M_1 simulates machine M_2 via simulation φ if whenever (η, Noise) is a trajectory of M_1 then $(\eta^*, \mathit{Noise}^*)$, defined by $\eta^*(\cdot, t) = \varphi^*(\eta(\cdot, t))$, is a trajectory of M_1. Here, Noise^* is computed by the residue operation (deleting isolated bursts) as in Definition 2.3. We will make, however, two refinements. First, we require the above condition only for those η for which the initial configuration $\eta(\cdot, 0)$ has been obtained by encoding, that is it has the form $\eta(\cdot, 0) = \varphi_*(\xi)$. Second, to avoid the transitional ambiguities in a history, we define the simulation decoding as a mapping Φ^* between *histories*, not just configurations: $\Phi^*(\eta, \mathit{Noise}) = (\eta^*, \mathit{Noise}^*)$.

Definition 2.6 (Simulation) Let M_1, M_2 be two generalized Turing machines, and let $\varphi_* : \mathrm{Configs}_{M_2} \to \mathrm{Configs}_{M_1}$ be a mapping from configurations of M_2 to those of M_1, such that it maps starting configurations into starting configurations. Let $\Phi^* : \mathrm{Histories}_{M_1} \to \mathrm{Histories}_{M_2}$ be a mapping. The pair (φ_*, Φ^*) is called a *simulation* (of M_2 by M_1) if for every trajectory (η, Noise) of

M_1 with initial configuration $\eta(\cdot, 0) = \varphi_*(\xi)$, the history $(\eta^*, Noise^*) = \Phi^*(\eta, Noise)$ is a trajectory of machine M_2. ⌟

In the noise-free case it is easy to find examples of simulations. However, in the cases with noise, finding any nontrivial example is a challenge, and depends on a careful definition of trajectories for generalized Turing machines.

2.8 Trajectories

(This completes the definition of the central concept of the proof—modulo the natural definitions spelled out in Section 3.4.) The set of trajectories of a generalized Turing machine M is defined in terms of constraints imposed on the fault-free parts of a history. We discuss these properties first informally.

Transition Function This property says (in more precise terms) that in a clean area, the transition function is obeyed.

The Spill Bound limits the extent to which a disordered interval can spread while the head is in it.

Escape limits the time for which the head can be trapped in a small area.

Attack Cleaning erodes the disorder as the head repeatedly enters and leaves it.

Pass Cleaning cleans the interior of an interval if the head passes over it enough times.

Interestingly, the Pass Cleaning property is only used in the proof of the other trajectory properties (and itself) for the simulated trajectory η^*. Without it, it could happen that the head slides over a large disordered area many times (staying inside) and, occasionally, extends it by new faults.

The definition below depends on the notions of current cell-pair, switch and dwell-period given in in Section 3.4, but should be understandable as it is.

Definition 2.7 Suppose that at times t' before a switching time t but after any previous switch, the current cell-pair has state $\mathbf{a}$, further after the switch the same cell-pair has state $\mathbf{a}'$ (including when one of these cells dies).

We say that the switch is *dictated by the transition function* if

$$(\mathbf{a}', \operatorname{sign}(\hat{h}_0(t') - \hat{h}_0(t))) = \tau(\mathbf{a}, \alpha),$$

where $\alpha = 0$ if the cell-pair is adjacent and 0 otherwise. ⌟

We will use the following constants:

$$c_{\text{spill}} = 2,\ c_{\text{marg}} = 13,\ c_{\text{pass}} = 39,\ c_{\text{esc}} = 19. \tag{2.5}$$

For an interval $I = [a, b)$ and some $c \geq 0$ let us define its *c-interior* $\operatorname{Int}(I, c) = [a + c, b - c)$.

Definition 2.8 (Trajectory) A history $(\eta, Noise)$ of a generalized Turing machine (2.4) with $\eta(t) = (A(t), h(t), \hat{\mathbf{h}}(t))$ is called a *trajectory* of M if the following conditions hold, in any noise-free space-time interval $I \times J$.

Transition Function Consider a switch, where the current cell-pair $\hat{\mathbf{h}}$ is inside the clean area, by a distance of at least $2.5B$. Then the new state of the current cell-pair, and the direction towards the new current head position are dictated by the transition function. If a new current cell has just been created then it is adjacent to the closer cell of the pair $\hat{\mathbf{h}}$. The only change on the tape occurs on the interval enclosing the new and old current cells. Further, the length of the dwell period before the switch is bounded by T.

Spill Bound An interval of disorder can spread by at most $c_{\text{spill}}B$ on either side while it contains the head.

Escape The head will leave any interval of size $\leq \lambda B$ with $1 \leq \lambda \leq 3\beta$ within time $c_{\text{esc}}\lambda^2 T$.

Attack Cleaning Suppose that the current cell-pair is at the end of a clean interval $[a, b)$ of size $\geq (c_{\text{marg}} + 1)B$, with the head at position x. Suppose further that the transition function directs the head right. Then by the time the head comes back to $x - c_{\text{marg}}B$, the clean interval extends at least to $[a, b + B/2)$. Similarly when "left" and "right" are interchanged.

Pass Cleaning Suppose that a path P makes at least π passes over interval I of size $\leq c_{\text{pass}}B$. Then at some time during P the interior $\text{Int}(I, c_{\text{marg}}B)$ of I becomes clean.

⌟

Our construction will set $\pi = 8k + O(1)$ for M_k, that is each 8 passes will raise the "organization level".

2.9 Scale-up

Above, we have set up the conceptual structure of the construction and the proof. Here are some of the parameters:

Definition 2.9 Let $Q_k = c_1 \cdot 2^{1.2^k}$, $U_k = Q_k^3 = c_1^3 \cdot 2^{3 \cdot 1.2^k}$, $\pi_k = 8k + c_2$ for appropriate constants $c_1, c_2 > 0$. These sequences clearly satisfy (2.3), and define $B_k = B_1 \prod_{i<k} Q_i$, $T_k = T_1 \prod_{i<k} U_i$. ⌟

What remains is the definition of the simulation program and the decoding Φ^*, and the proof that with this program, the properties of a trajectory $(\eta, Noise)$ of machine $M = M_k$ imply that the history $\Phi^*(\eta, Noise) = (\eta^*, Noise^*)$ obeys the same trajectory requirements on the next level. The program is described in Sections 4-6. The most combinatorially complex part of the proof of trajectory properties is in the key Section 8, proving the trajectory properties related to bounding and eliminating disorder. Section 9 wraps up the proof of the main theorem. (Some parts of the paper will get added detail in the final version, but the present version leaves no essential question unanswered.)

3 Some formal details

We give here some details that were postponed from the overview section.

3.1 Examples

The following examples show the difficulties responsible for various complexities of the construction and proof. Some of them may not be completely understandable without the details of the following program. You can skip these examples safely, and return to them later when wondering about the motivation for some feature.

Example 3.1 (Need for zigging) When the head works in a colony of machine M_1 performing a simulation of a cell of machine M_2, it works in sweeps across the colony. But a small burst of faults could reverse the head in the middle of a sweep, leaving it uncompleted. This way local faults could create non-local inconsistency. ⌟

To handle the problem of Example 3.1, the head will proceed in zigzags: every step advancing the head in the simulation is followed by Z steps of going backward and forward again, just checking consistency (and starting a healing process if necessary). The parameter Z will be chosen appropriately.

Example 3.2 (Need for feathering) Some big noise can create a number of intervals $I_1, I_2, \ldots, I_n$ consisting of colonies of machine M_1, each interval with its own simulated head, where the neighboring intervals are in no relation to each other. When the head is about to return from the end of I_k (never even to zig beyond it, see the discussion after Example 3.1), a small burst of faults can carry it over to I_{k+1} where the situation may be symmetric: it will continue the simulation that I_{k+1} is performing. (The rightmost colony of I_k and the leftmost colony of I_{k+1} need not be complete: what matters is only that the simulation in I_k would not bring the head beyond its right end, and the simulation in I_{k+1} would not bring the head beyond its left end.)

The head can be similarly captured to I_{k+2}, then much later back from I_{k+1} to I_k, and so on. This way the restoration of structure in M_2 may be delayed too long. ⌟

The device by which we will mitigate the effect of this kind of capturing is another property of the the movement of the head which will call *feathering*: if the head turns back from a tape cell then next time it must go beyond. This requires a number of adjustments to the program (see later).

Example 3.3 (Two slides over disorder) This example shows the possibility for the head to slide twice over disorder without cleaning it.

Consider two levels of simulation as outlined in Section 2.2: machine M_1 simulates M_2 which simulates M_3. The tape of M_1 is subdivided into colonies of size Q_1. A burst of faults on level 1 has size $O(1)$, while a burst on level 2 has size $O(Q_1)$.

Suppose that M_1 is performing a simulation in colony C_0. An earlier big burst may have created a large interval D of disorder on the right of C_0, even reaching into C_0. For the moment, let C_0 be called a *victim* colony. Assume that the left edge of D represents the last stage of a transfer operation to the right neighbor colony $C_0 + Q_1$. When the head, while performing its work in C_0, moves close to its right end, a small burst may carry it over into D. There it will be "captured", and continue the (unintended) right transfer operation. This can carry the head, over several successful colony simulations in D, to some victim colony C_1 on the right from which it will be captured to the right similarly. This can continue over new and new victim colonies C_i (with enough space between them to allow for new faults to occur), all the way inside the disorder D. So the M_2 cells in D will fail to simulate M_3.

After a while the head may return to the left in D (performing the simulations in its colonies). When it gets at the right end of a victim colony C_i, a burst of faults might move it back there. There is a case when C_i now can just continue its simulation and then send the head further left: when before the head was captured on its right, it was in the last stage of simulating a left turn of the head of machine M_2.

In summary, a big burst of faults can create a disordered area D which can capture the head and on which the head can slide forward and back without recreating any level of organization beyond the second one. ⌟

Example 3.4 (Many slides over disorder) Let us describe a certain "organization" of a disordered area in which an unbounded number of passes may be required to restore order. For some $n < 0$, let the cells of M_1 at positions $x_{-Q_1}, \ldots, x_n$, where $x_{i+1} = x_i + B_1$, represent part of a healthy colony $C(x_{-Q_1})$ starting at x_{-Q_1}, where x_n is the rightmost cell of $C(x_{-Q_1})$ to which the head would come in the last sweep before the simulation will move to the *left* neighbor colony $C(x_{-2Q_1})$. Let them be followed by cells $x_{n+1}, \ldots, x_{Q_1-1}, \ldots$ which represent the last sweep of a transfer operation to the *right* neighbor colony $C(x_0)$. If the head is in cell x_n, a burst of faults can transfer it to x_{n+1}. The cell state of M_2 simulated by $C(x_{-Q_1})$ need to be in *no relation* to the cell state of M_2 simulated by $C(x_0)$. This was a capture of the head by a burst of M_1 across the point 0, to the right.

We can repeat the capture scenario, say around points iQ_1Q_2 for $i = 1, 2, \ldots$, and this way cells of M_3 simulated by M_2 (simulated by M_1) can be defined arbitrarily, with no consistency needed between any two neighbors. (We did not write iQ_1 just in case bursts are not allowed in neighboring colonies.) In particular, we can define them to implement a *leftward* capture scenario via level 3 bursts at points $iQ_1Q_2Q_3Q_4$, allowing to simulate arbitrary cells of M_5 with no consistency requirement between neighbors. So M_5 could again implement a rightward capture scenario, and so on. In summary, a malicious arrangement of disorder and noise allows k passes after which the level of organization is still limited to level $2k + 1$. ⌟

Our construction will ensure that, on the other hand, $O(k)$ passes (free of k-level will restore organization to level k. This property of the construction will be incorporated into our definition of a generalized Turing machines as the "magical" property (C) above.

3.2 Codes

The input of our computation will be encoded by some error-correcting code, to defend against the possibility of losing information even at the first reading.

Definition 3.5 (Codes) Let Σ_1, Σ_2 be two finite alphabets. A *block code* is given by a positive integer Q—called the *block size*—and a pair of functions

$$\psi_* : \Sigma_2 \to \Sigma_1^Q, \quad \psi^* : \Sigma_1^Q \to \Sigma_2$$

with the property $\psi^*(\psi_*(x)) = x$. Here ψ_* is the encoding function (possibly introducing redundancy) and ψ^* is the decoding function (possibly correcting errors). The code is extended to (finite or infinite) strings by encoding each letter individually:

$$\psi_*(x_1, \ldots, x_n) = \psi_*(x_1) \cdots \psi_*(x_n).$$

⌟

3.3 Proof of the sparsity lemma

Proof of Lemma 2.4. Rectangle $B(\mathbf{x}, (B_1, T_1))$ has size 1, the probability of our event is $< \varepsilon$. Let us prove the inequality by induction, for $k+1$. Let

$$p_k = \varepsilon \cdot 2^{-1.5^{k-1}}.$$

Suppose $\mathbf{y} \in E^{(k)} \cap B(\mathbf{x}, (B_{k+1}, T_{k+1}))$. This event depends at most on the rectangle $B(\mathbf{x}, \gamma(B_k, T_k))$. Then, according to the definition of $E^{(k)}$, there is a point

$$\mathbf{z} \in B(\mathbf{y}, \gamma_k(B_{k+1}, T_{k+1})) \cap E^{(k)} \setminus B(\mathbf{y}, \beta(B_k, T_k)). \tag{3.1}$$

Consider a standard partition of the (two-dimensional) space-time into rectangles $K_p = \mathbf{c}_p + [-B_k, B_k) \times [-T_k, T_k)$ with centers $\mathbf{c}_1, \mathbf{c}_2, \ldots$. The rectangles K_i, K_j containing $\mathbf{y}$ and $\mathbf{z}$ respectively intersect $B(\mathbf{x}, 2\gamma_k(B_{k+1}, T_{k+1}))$. The triple-size rectangles $K_i' = c_i + [-3B_k, 3B_k) \times [-3T_k, 3T_k)$ and K_j' are disjoint, since the lower bound on β in Definition 2.3 and (3.1) imply $|y_1 - z_1| > \beta B_k$ and $|y_2 - z_2| > \beta T_k$. The set $E^{(k)}$ must intersect two rectangles K_i, K_j of size $2(B_k, T_k)$ separated by at least $4(B_k, T_k)$, of the big rectangle $B(\mathbf{x}, 2\gamma_k(B_{k+1}, T_{k+1}))$.

By the inductive hypothesis, the event $\mathcal{F}_i$ that K_i intersects E_k has probability bound p_k. It is independent of the event $\mathcal{F}_j$, since these events depend only on the triple size disjoint rectangles K_i' and K_j'. The probability that both of these events hold is at most p_k^2. The number of possible rectangles K_p intersecting $B(\mathbf{x}, 2\gamma_k(B_{k+1}, T_{k+1}))$ is at most $C_k := ((2\gamma_k^2 U_k Q_k) + 2)^2$, so the number of possible pairs of rectangles is at most $C_k^2/2$, bounding the probability of our event by

$$\begin{aligned} C_k^2 p_k^2/2 &= C_k^2 \cdot \varepsilon^2 2^{-1.5^k} \cdot 2^{-0.5 \cdot 1.5^{k-1}} \\ &= \varepsilon 2^{-1.5^k} \cdot \varepsilon C_k^2 2^{-0.5 \cdot 1.5^{k-1}}. \end{aligned}$$

Since $\lim_k \frac{\log U_k}{1.5^k} = 0$, the last factor is ≤ 1 for sufficiently small ε. □

3.4 Configuration, history

A configuration, as defined below, contains a pair of positions $\hat{\mathbf{h}} = (\hat{h}_0, \hat{h}_1)$ called the *current cell-pair*: In difference to the Turing machines of Section 2.6, the position of the head may not be exactly between the current cells: this allows the model to fit into the framework where a generalized Turing machine M^* is simulated by some (possibly generalized) Turing machine M. The head h of M—made equal to that of M^*—may oscillate inside and around the current cell-pair of M^*.

Definition 3.6 (Configuration) A *configuration* ξ of a generalized Turing machine (2.4) is a tuple

$$(A, h, \hat{\mathbf{h}}) = (\xi.\text{tape}, \xi.\text{pos}, (\xi.\text{cur-cell}_0, \xi.\text{cur-cell}_1))$$

where $A : \mathbb{Z} \to \Sigma$ is the tape, $h \in \mathbb{Z}$ is the head position, $\hat{\mathbf{h}} \in \mathbb{Z}^2$ is the current cell-pair. We have $A(p) = \text{Vac}$ in all but finitely many positions p. Whenever the interval $h + [4B, 4B)$ is clean the current cell-pair must be within it. Let

$$\text{Configs}_M$$

denote the set of all possible configurations of a Turing machine M. ⌟

All the above definitions can clearly be localized to define a configuration over a space interval I containing the head.

Definition 3.7 (History) For a generalized Turing machine (2.4), consider a sequence $\eta = (\eta(0), \eta(1), \dots)$, along with a noise set *Noise*. The pair (η, Noise) will be called a *history* of machine M if the following conditions hold.

- $|h(t) - h(t')| \leq |t' - t|$.
- In two consecutive configurations, the content $A(p, t)$ of the positions p not in $h(t) + [-2B, 2B)$ remains the same.
- At each noise-free switching time the head is on the new current cell-pair: $\hat{h}_1(t) = h(t)$. (In particular, when at a switching time a current cell becomes Vac, the head must already be elsewhere.)
- The length of any noise-free dwell period in which the head is staying on clean positions is at most T.

We say that a cell *dies* in a history if it becomes Vac. Let

$$\text{Histories}_M$$

denote the set of all possible histories of M. ⌟

The above definition can be *localized* to define a history *over a space-time rectangle* $I \times J$ containing the head.

3.5 Hierarchical codes

Recall the notion of a code in Definition 3.5.

Definition 3.8 (Code on configurations) Consider two generalized Turing machines M_1, M_2 with the corresponding alphabets and transition functions, where B_2/B_1 is an integer denoted $Q = Q_1$. Assume that a block code $\psi_* : \Sigma_2 \to \Sigma_1^Q$ is given, with an appropriate decoding function, ψ^*. Symbol $a \in \Sigma_2$, is interpreted as the content of some tape square.

This block code gives rise to a *code on configurations*, that is a pair of functions

$$\varphi_* : \text{Configs}_{M_2} \to \text{Configs}_{M_1}, \quad \varphi^* : \text{Configs}_{M_1} \to \text{Configs}_{M_2}$$

that encodes some (initial) configurations ξ of M_2 into configurations of M_1: each cell of M_2 is encoded into a colony of M_1 occupying the same interval. Formally, assuming $\xi.\text{cur-cell}_j = \xi.\text{pos} + (j-1)B_2$, $j = 0, 1$ we set $\varphi_*(\xi).\text{pos} = \xi.\text{pos}$, $\varphi_*.\text{cur-cell}_j = \varphi_*(\xi).\text{pos} + (j-1)B_2$, and

$$\varphi_*(\xi).\text{tape}(iB_2, \ldots, (i+1)B_2 - B_1) = \psi_*(\xi.\text{tape}(i)).$$

A configuration ξ is called a *code configuration* if it has the form $\xi = \varphi_*(\zeta)$. ⌟

Definition 3.9 (Hierarchical code) For $k \geq 1$, let Σ_k be an alphabet, of a generalized Turing machine M_k. Let $Q_k = B_{k+1}/B_k$ be an integer (viewed as colony size), let φ_k be a code on configurations defined by a block code

$$\psi_k : \Sigma_{k+1} \to \Sigma_k^{Q_k}$$

as in Definition 3.8. The sequence (Σ_k, φ_k), $(k \geq 1)$, is called a *hierarchical code*. For this hierarchical code, configuration ξ^1 of M_1 is called a *hierarchical code configuration* of height k if a sequence of configurations $\xi^2, \xi^3, \ldots, \xi^k$ of $M_2, M_3, \ldots, M^k$ exists with

$$\xi^i = \varphi_{*i}(\xi^{i+1})$$

for all i. If we are also given a sequence of mappings Φ_1^*, Φ_2^*, ... such that for each i, the pair (φ_{i*}, Φ_i^*), is a simulation of M_{i+1} by M_i then we have a *hierarchy of simulations* of height k. ⌟

We will construct a hierarchy of simulations whose height grows during the computation—by a mechanism to be described later.

4 Simulation structure

In what follows we will describe the program of the reliable Turing machine: a hierarchical simulation of each M_{k+1} by M_k, with an added mechanism to raise the height of the hierarchy when needed. (Note that this is just one program, using a parameter k.) Most of the time, we will write

$M = M_k$, $M^* = M_{k+1}$. Cells will be grouped into colonies of size $Q = B^*/B$. Simulating one step of M^* takes a sequence of steps of M constituting a *work period*. Machine M will perform the simulation as long as the noise in which it operates is $(\beta(B,T), \gamma(B^*,T^*))$-sparse— meaning, informally, that a burst of noise affects at most β consecutive tape cells, and there is at most one burst in any γ neighboring work periods. A design goal for the program is to correct a burst within space much smaller than a colony.

The presentation is not entirely modular in the sense that after the definition of each part it is shown what properties of the simulation it guarantees. This is since noise occurring during any part of the program can change the environment of the head into some state corresponding to an arbitrary other part.

Modes were introduced in Section 4.2. Ordinary simulation proceeds in the normal mode. To see whether consistency, that is the basic structure supporting this process, is broken somewhere, each step will check whether the current cell-pair is *coordinated* (see Definition 5.8). If not then the mode of the current pair will be changed into *healing*. We will also say that *alarm* will be called. On the other hand, the state may enter into *rebuilding* mode on some indications that healing fails.

4.1 Error-correcting code

Let us add error-correcting features to the block codes introduced in Definition 3.5.

Definition 4.1 (Error-correcting code) A block code is (β,t)-*burst-error-correcting*, if for all $x \in \Sigma_2$, $y \in \Sigma_1^Q$ we have $\psi^*(y) = x$ whenever y differs from $\psi_*(x)$ in at most t intervals of size $\leq \beta$. For such a code, we will say that a word $y \in \Sigma_1^Q$ is *r-compliant* if it differs from a codeword of the code by at most r intervals of size $\leq \beta$. ⌟

Example 4.2 (Repetition code) Suppose that $Q \geq 3\beta$ is divisible by 3, $\Sigma_2 = \Sigma_1^{Q/3}$, $\psi_*(x) = xxx$. If $y = y(1)\dots y(Q)$, then $x = \psi^*(y)$ is defined by $x(i) = \mathrm{maj}(y(i), y(i+Q/3), y+2Q/3)$. For all $\beta \leq Q/3$, this is a $(\beta,1)$-burst-error-correcting code. If we repeat 5 times instead of 3, we get a $(\beta,2)$-burst-error-correcting code. ⌟

Example 4.3 (Reed-Solomon code) There are much more efficient such codes than just repetition. One, based on the Reed-Solomon code, is outline in Example 4.6 of [6]. If each symbol of the code has l bits then the code can be up to 2^l symbols long. Only $2t\beta$ of its symbols need to be redundant in order to correct t faults of length β. ⌟

Consider a (generalized) Turing machine (Σ,τ) simulating some Turing machine (Σ^*,τ^*). We will assume that the alphabet Σ^* is a subset of the set of binary strings $\{0,1\}^l$ for some $l < Q$.

Definition 4.4 (Interior) Let

PadLen

be a parameter to be defined later (in (4.3)). Consider an interval I of neighbor cells. A cell belongs to the *interior* of I if there are at least *PadLen* neighbors between it and the complement of I. In particular, we will talk about the interior of a colony. ⌟

We will store the coded information in the interior of the colony, since it is more exposed to errors near the boundaries. So let (v_*, v^*) be a $(\beta, 2)$-burst-error-correcting block code with

$$v_* : \{0,1\}^l \cup \{\emptyset\} \to \{0,1\}^{(Q-2\cdot PadLen)B}.$$

We could use, for example, the repetition code of Example 4.2. Other codes are also appropriate, but we require that there are some fixed Turing machines Encode and Decode computing them:

$$v_*(x) = \text{Encode}(x), \quad v^*(y) = \text{Decode}(y).$$

Also, these programs must work in quadratic time and linear space on a one-tape Turing machine (as the repetition code certainly does).

Let us now define the block code (ψ_*, ψ^*) used below in the definition of the configuration code (φ_*, φ^*) outlined in Section 3.5:

$$\psi_*(a) = 0^{PadLen} v_*(a) 0^{PadLen}. \tag{4.1}$$

The decoded value $\psi^*(x)$ is obtained by first removing *PadLen* symbols from both ends of x to get x', and then computing $v^*(x')$. It will be easy to compute the configuration code from ψ_*, once we know what fields there need initialization.

4.2 Rule language

The generalized Turing machines M_k to be defined differ only in the parameter k. We will denote therefore M_k frequently simply by M, and M_{k+1}, simulated by M_k, by M^*. Similarly we will denote the colony size Q_k by Q.

We will describe the transition function $\tau_k = \tau$ mostly in an informal way, as procedures of a program, these descriptions are readily translatable into a set of *rules*. Each rule consists of some (nested) conditional statements, similar to the ones seen in an ordinary program: "**if** *condition* **then** *instruction* **else** *instruction*", where the condition is testing values of some fields of the observed cell-pair, and the instruction can either be elementary, or itself a conditional statement. The elementary instructions are an *assignment* of a value to a field of a cell symbol, or a command to move the head. It will then be possible to write one fixed *interpreter* Turing machine that carries out these rules.

Assignment of value x to a field y of the state or cell symbol will be denoted by $y \leftarrow x$.

4.3 Fields

In what follows we describe some of the most important fields we will use in the cells; others will be introduced later.

A properly formatted configuration of M splits the tape into blocks of Q consecutive cells called *colonies*. One colony of the tape of the simulating machine represents one cell of the simulated machine. The two colonies that correspond to the cell-pair that the simulated machine is scanning is called the *base colony-pair* (a precise definition will refer to the actual history of the work of M). Sometimes the left base colony will just be called the *base colony*. Most of the computation proceeds over the base colony-pair. The direction of the simulated head movement, once figured out by the computation, is called the *drift*. Two neighbor colonies may not be adjacent, in which case the cells will form a *bridge* between them.

Here is a description of some of the fields:

Mode The present behavior of the computation will be characterized by a field of the current cell called the *mode*:

$$Mode \in \{\text{Normal}, \text{Healing}, \text{Rebuilding}, \text{Booting}\}.$$

If its value is Normal we will say that the computation is in *normal* mode. In this case, the machine is engaged in the regular business of simulation. The *healing* mode tries to correct some local fault due to a couple of neighboring bursts of faults (of size β), while the *rebuilding* mode attempts to restore the colony structure on the scale of a couple of colonies. The *booting* mode is used for setting up the structure initially and for starting or continuing the simulation of the Turing machine G of the main theorem.

Info The *Info* track of a colony of M contains the string that encodes the content of the simulated cell of M^*.

Address The field *Addr* of the cell shows the position of the cell in its colony: it takes values in $[0, Q)$.

Drift The direction in $\{-1, 1\}$ in which the simulated head moves will be recorded on the track *Drift*.

Sweep The *Sweep* field keeps track of the number of sweeps that the head made during the work period.

Kind Cells will be designated as belonging to a number of possible *kinds*, signaled by the field *Kind* with values Booting, Stem, Member_0, Member_1, Bridge. Here is a description of their role.

- A cell is of the Booting kind if it is on the top level of simulation (see Section 2.2).
- Stem is the kind of newly created cells whose role in the colony structure is not clear yet, or of cells whose previous role has been erased.

- Cells of the base colony-pair are of type Member_0 and Member_1 respectively. If the base colony-pair is not adjacent then there will be a *bridge* of $< Q$ adjacent cells of type Bridge between them. We will say that the bridge *extends* a colony if its cells are adjacent to it.

Under normal conditions, if the *Drift* track on the bridge has value -1 then it is extending the left base colony, otherwise is extending the right base colony. Cells of the bridge will continue the addressing (modulo Q) of the colony it extends.

Heal, Rebuild During healing, some special fields of the state and cell are used, treated as subfields of the field *Heal*. In particular, there will be a *Heal.Sweep* field. During rebuilding, we will work with subfields of the field *Rebuild*. A cell will be called *marked for rebuilding* if *Rebuild.Sweep* $\neq 0$.

4.4 Head movement

The global structure of a work period is this:

Simulation phase Compute the new state of the simulated cell-pair, and the simulated direction (called the drift). Then check the "meaningfulness" of the result.

Transfer phase The head moves into the neighbor colony-pair in the simulated head direction called drift (building or rebuilding a bridge if needed). Redundancy is used to protect this movement from faults.

During the work period the head sweeps, roughly, back-and-forth between the ends of the base colony-pair. The field *Sweep* keeps track of the number of sweeps made within the work period. This together with the address and kind of the current cell is used by the program to determine the actions to be performed.

Definition 4.5 (Front) As the head sweeps in a certain direction, it increases *Sweep* field by 1. The last site in which this increase occurred will be called the *front*. ⌟

Globally in a configuration, due to earlier faults, there may be more than one front, but locally we can talk about "the" front without fear of confusion. The direction of the sweep s is determined by its parity:

$$\text{dir}(s) = (-1)^{s+1}.$$

Due to the zigging and feathering requirements mentioned in Section 2.4, there will be two kinds of turns: *small* turns and *big* turns. The process uses the parameters

$$f = 6,\ Z = \pi^{1+\rho},\ F = Z\pi^{2+\rho} \tag{4.2}$$

with $\rho > 0$, which define the earlier introduced parameter

$$\mathit{PadLen} = F \log Q. \tag{4.3}$$

The choices will be motivated in Sections 4.4.2 and 8.3.

4.4.1 Zigging

A *zigzag* movement will be superimposed on sweeping, so that the head can check the consistency of a few cells around the front. The process creates a *frontier zone* of about $2Z$ cells in both directions around the front, where Z was defined in (4.2). This interval is marked by the values of the field

$$ZigAddr \in [-2Z, 2Z)$$

which is undefined elsewhere. After every f steps of progress, the head will perform a forward-backward-forward zigzag checking and updating the zone. For this it uses *small turns* defined in Section 4.4.2, so it may need a few more steps to find a turning point. Due to the spacing of all turns, this will need no more than f steps under normal conditions.

Remarks 4.6 1. The need for backward zigging was explained in Example 3.1.

2. The forward-zigging property and the frontier zone will be used in the proof of Lemma 8.14. We will need to show there that in the absence of new noise, after a constant number of passes, a clean interval J of size, say, $\geq 6ZB$ will extend to the right until it reaches the end of a colony-pair or of a rebuilding area. Forward zigging will prevent any disorder from turning the head back prematurely.
3. The choice of Z in (4.2) will be justified in the same proof; it also lower-bounds the size of new noise capable of turning back the head.

⌟

4.4.2 Feathering

Example 3.2 above indicated the need for the following feature of the simulation program.

Definition 4.7 (Feathering) A Turing machine execution is said to have the *c-feathering* property if the following holds. If the head turned back at a position x at some time, then next time it comes within distance c of x, it can turn back only at least c steps beyond x. ⌟

(The name "feathering" refers to the picture of the path of the head in a space-time diagram.) The following example suggests that any computation can be reorganized to accomodate feathering, without too much extra cost.

Example 4.8 (1-feathering) Suppose that, arriving from the left at position 1, the head decides to turn left again. In repeated instances, it can then turn back at the following sequence of positions:

$$1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, \ldots$$

⌟

If in the original execution the head turned back t consecutive times to the left from position p, then now it will turn back from somewhere in a zone of size $O(\log t)$ to the right of p in each of these times. (Computing the exact turning point is not necessary.) To remember a turn, a field

Turned,

whose default value is 0, will be set to 1. When a cell is passed without turning, we reset *Turned* to 0. Moreover, a "big turn" (see below) will be allowed only in a distance at least F from the just passed place of a previous big turn.

Definition 4.9 (Digression) Whenever a turn is postponed since the head is not allowed to turn due to feathering, the simulation to be carried out by the head is suspended until the head returns. This is called a *digression*. ⌟

Small turn Small turns will be done at zigging in during normal and healing mode (see later). Before turning, the head moves ahead at least f steps, then moves more until it finds a place with $Turned = 0$. If one is not found within $2f$ steps then call alarm but still don't turn. Such an event will be called *small turn starvation*.

Big turn Big turns will be done during booting, and the ends of normal and rebuilding sweeps. They are governed with the help of a pair of fields called $FCount_1, FCount_2$ with default value 0. Consider a big turn to the left (the other direction is analogous). The place of big turn will be marked by a segment (called a *train*) of $Z/2$ cells, having $FCount_1 \geq F$, immediately to the left of the place of turn. When a big turn is initiated, a train is set up with $FCount_j = 1$, $j = 1, 2$ in the frontier zone. Then in consecutive zigging, it is moved to the right with the frontier zone, leaving behind cells with $FCount_j = 0$, and increasing its $FCount_j$ by 1. When a value $\geq F$ is reached having the cell ahead $Turned = 0$, the big left turn will be made. The train with $FCount_1 > 0$ that is left there will be called the *footprint* of the big left turn. If a segment of $> 2E$ cells with $FCount_1 > 0$ is encountered before the above conditions allow a turn, then the train is reset to $FCount_1 = 1$ and continues forward (the value $FCount_2$ is not reset). If $FCount_2 = 2Q$ is reached then start (or restart) rebulding, and set $FCount = 1$ but still don't turn: such an event will be called a *big turn starvation*.

Note that a big turn cannot happen without passing back through a train with $FCount_1 \geq F$.

Definition 4.10 A clean interval will be called *safe for turns* if every sequence of $2f$ cells in it contains a cell with $Turned = 0$ and it has no sequence of more than $3F \log Q$ footprints of a big turn closer than $2F$ cells to each other. ⌟

An interval rewritten by noise can, of course, be unsafe for both small and big turns, even if it is clean. But our machine M will have the property that when a fault-free path passes over a clean interval then the interval becomes safe for turns. We give here only an informal argument; formal proof must wait until a complete definition of the simulation. Zigs and the turns during healing are by the definition spaced by $\geq f$ cells apart, so there will be at most 2 cells with $Turned = 1$

in any interval of size f contributed by zigging and at most 2 contributed by healing. As for big turns, as indicated in Example 4.8 (for $F = 1$), a big turn attempt will be delayed by at most F times the logarithm of the total number of big turns inside a colony or a rebuild area.

The simulated Turing machine will also have the feathering property, therefore the simulation will not turn back from one and the same colony repeatedly, without having passed it in the meantime.

Remark 4.11 The size of the parameter F is motivated by the proof of Lemma 8.14. Here is a sketch of the argument (it can be safely skipped now). At some time t_0 in some interval I we will have clean subintervals $J_k(t_0)$, $k = 1, 2, \ldots$ of size $\geq 6ZB$ in which no fault will appear, and which are separated from each other by areas of size $O(\pi^2 ZB) \ll FB$. For times $t > t_0$ we will track the maximal clean intervals $J_k(t)$ containing the middle of $J_k(t_0)$.

Assume that the head passes over I noiselessly left to right and later also noiselessly from right to left. If the head moves in a zigging way to the right then the Attack Cleaning property will clean out the area between $J_i(t)$ and $J_{i+1}(t)$, joining them. This does not happen only in case of a programmed turn from the right end of $J_i(t)$, in the course of the simulation or rebuilding. But then in the next pair of passes over I, the feathering property implies that the programmed turn from the end of $J_i(t)$ is at least a distance FB to the right. Our choice of F implies that then $J_i(t)$ will be joined to $J_{i+1}(t)$. So two noise-free passes would join all the intervals $J_i(t)$ into a clean area. ⌟

4.5 Simulation phase

The simulation phase, called the `Compute` rule, computes new values for current cell-pair of the simulated machine M^* represented by the current (base) colony-pair, and the direction of the move of the head of M^*. During the execution of this rule, the head sweeps the base colony-pair. The cell state of M^* will be stored on the track *Info* of the representing colony. The move direction of M^* will be written into the *Drift* field of *each* cell of the base colony-pair (so the whole track must be filled with the same symbol $d \in \{-1, 1\}$.

The rule `Compute` relies on some fixed $(\beta, 3)$ burst-error-correcting code, moreover it expects each of the words found on the *Info* track to be 2-compliant (Definition 4.1). There is a rule `ComplianceCheck` to check whether a word is 2-compliant.

The rule `Compute` essentially repeats 3 times the following *stages*: decoding, applying the transition, encoding. Then it calls `ComplianceCheck`; if the latter fails it will mark the colony for rebuilding. It uses some additional tracks like *Work* for most of the computation. The *Info* track will not be modified before all the *Hold*$[j]$ tracks are written.

In more detail:

1. For $j = 1, \ldots, 3$ do
   a. Calling by $\mathbf{a}$ the pair of strings found on the *Info* track of the interiors of the base colonies, decode it into the pair of strings $\tilde{\mathbf{a}} = v^*(\mathbf{a})$ (the current state of the simulated cell-pair), and

store it on some auxiliary track in the base colony-pair. Do this by simulating the universal machine: $\tilde{\mathbf{a}} = \text{Decode}(\mathbf{a})$.

Each cell stores a track segment of the tape of machine Decode. When the head of M is on the cell with the simulated head of Decode—let us call this the *head segment*—then one step of M simulates a number of steps of Decode as large as the size of this segment or until the simulated head leaves the segment (thus changing the head segment).

For simplicity of analysis, each pair of sweeps changes only on the head segment (and possibly the neighbor that becomes the new head segment). We accept this slowdown now.

b. Compute $(\mathbf{a}', d) = \tau^*(\tilde{\mathbf{a}}, \alpha)$, where $\alpha = 1$ if the pair of colonies is adjacent, else 0. Since the program of the transition function τ^* is not written explicitly anywhere, this "self-simulation" step is discussed in detail in Section 4.6.

c. Write the encoded new cell states $v_*(\mathbf{a}')$ onto the $Hold[j].Info$ track of the interior of the base colony-pair. Write d into the $Hold[j].Drift$ field of *each cell* of the left base colony.

If the new state a'_i is a vacant one for $i = 0$ or 1, that is $a'_i.Kind^* = \text{Vac}^*$, then write 1 onto the $Hold[j].Doomed$ track of the corresponding (left or right) base colony—else write 0.

2. Sweeping through the base colony-pair, at each cell compute the majority of $Hold[j].Info$, $j = 1, \ldots, 3$, and write it into the field *Info*. Proceed similarly, and simultaneously, with *Drift*.

3. For $j = 1, \ldots, 3$, call `ComplianceCheck` on the *Info* track, and write the resulting bit into the $Compliant_j$ track.

   Then pass through the colony and turn each cell in which the majority of $Compliant_j$, $j = 1, \ldots, 3$ is false, into a stem cell (thus destroying the colony-pair if the result was false everywhere).

4. If the *Output* field of the simulated cell is defined, write it into the output field of the left end-cell of each colony.

Part 4 achieves that when the computation finishes on some simulated machine M_k, its output value in cell 0 of M_k will "trickle down" to the output field of cell 0 of M_1.

As seen in this construction one can find a constant c_{redund} with the property

$$s_{k+1} \le Q_k s_k \le c_{\text{redund}} s_{k+1} \tag{4.4}$$

(see the definition of s_k in (2.1)), that is a colony of M_k represents a cell of M_{k+1} with redundancy factor c_{redund}. The number of steps U_k taken by M_k needed for one work period of simulation is $O(Q_k^2)$.

Remark 4.12 There is a mechanism more economical on storage, without the full-width *Work* and $Hold[j]$ tracks but with some added complexity, see Section 9.2. This allows a space redundancy factor $1 + \delta_k$ with $\prod_k (1 + \delta_k) < \infty$, yielding a constant space redundancy factor for the whole hierarchy. ⌟

4.6 Self-simulation

Let us elaborate step 1b of Section 4.5.

4.6.1 New primitives

The simulation phase makes use of the tracks *Work* mentioned above, and the track

$$\mathit{Index}$$

that can store a certain address of a colony.

Recall from Section 4.2 that the program of our machine is a list of nested "**if** *condition* **then** *instruction* **else** *instruction*" statements. As such, it can be represented as a binary string

$$R.$$

If one writes out all details of the construction of the present paper, this string R becomes explicit, an absolute constant. But in the reasoning below, we treat it as a parameter.

Let us provide a couple of *extra primitives* to the rules. First, they have access to the parameter k of machine $M = M_k$, to define the transition function

$$\tau_{R,k}(\mathbf{a}).$$

The other, more important, new primitive is a special instruction

$$\texttt{WriteProgramBit}$$

in the rules. When called, this instruction makes the assignment $\mathit{Work} \leftarrow R(\mathit{Index})$. This is the key to self-simulation: *the program has access to its own bits*. If $\mathit{Index} = i$ then it writes $R(i)$ onto the current position of the *Work* track.

4.6.2 Simulating the rules

The structure of all rules is simple enough that they can be read and interpreted by a Turing machine in reasonable time:

Theorem 2 *There is a Turing machine* Interpr *with the property that for all positive integers k, string R that is a sequence of rules, and a pair of bit strings $\mathbf{a} = (a_0, a_1)$ with $a_j \in \Sigma_k$,*

$$\text{Interpr}(R, 0^k, \mathbf{a}) = \tau_{R,k}(\mathbf{a}).$$

The computation on Interpr *takes time $O(|R| \cdot |\mathbf{a}|)$.*

The proof parses and implements the rules in the string R; each of these rules checks and writes a constant number of fields.

Implementing the `WriteProgramBit` instruction is straightforward: Machine Interpr determines the number i represented by the simulated *Index* field, looks up $R(i)$ in R, and writes it into the simulated *Work* field.

There is no circularity in these definitions:

- The instruction `WriteProgramBit` is written *literally* in R in the appropriate place, as "`WriteProgramBit`". The string R is *not part* of the rules (that is of itself).
- On the other hand, the computation in Interpr$(R, 0^k, \mathbf{a})$ has *explicit* access to the string R as one of the inputs.

Let us show the computation step invoking the "self-simulation" in detail. In the earlier outline, step 1b of Section 4.5 said to compute $\tau^*(\tilde{\mathbf{a}})$ (for the present discussion, we will just consider computing $\tau^*(\mathbf{a}) = \tau_{k+1}(\mathbf{a})$), where $\tau = \tau_k$, and it is assumed that $\mathbf{a}$ is available on an appropriate auxiliary track. We give more detail now of how to implement this step:

1. Onto the *Work* track, write the string R. To do this, for *Index* running from 1 to $|R|$, execute the instruction `WriteProgramBit` and move right. Now, on the *Work* track, add 0^{k+1} and $\mathbf{a}$. String 0^{k+1} can be written since the parameter k is available. String $\mathbf{a}$ is available on the track where it is stored.
2. Simulate the machine Interpr on track *Work*, computing $\tau_{R,k+1}(\mathbf{a})$.

This implements the forced self-simulation. Note what we achieved:

- On level 1, the transition function $\tau_{R,1}(\mathbf{a})$ is defined completely when the rule string R is given. It has the forced simulation property by definition, and string R is *"hard-wired"* into it in the following way. If $(\mathbf{a}', d) = \tau_{R,1}(\mathbf{a})$, then

$$a'_0.Work = R(a_0.Index)$$

  whenever $a_0.Index$ represents a number between 1 and $|R|$, and the values $a_0.Sweep$, $a_0.Addr$ satisfy the conditions under which the instruction `WriteProgramBit` is called in the rules (written in R).
- The forced simulation property of the *simulated* transition function $\tau_{R,k+1}(\cdot)$ is achieved by the above defined computation step—which relies on the forced simulation property of $\tau_{R,k}(\cdot)$.

Remark 4.13 This construction resembles the proof of Kleene's fixed-point theorem, and even more some self-reproducing programs (like a program p in the language C causing the computer to write out the string p). ⌟

4.7 Transfer phase

In the transfer phase the `Transfer` rule takes over: control will be transferred to the neighbor colony-pair in the direction of the simulated head movement which we called the *drift*. The phase takes only a constant number of sweeps. During this phase, the range of the head includes the base colony-pair and a neighbor colony in the direction of the drift called *target colony*, including possible bridges between them. The bridges present some extra complication—let us address it.

Definition 4.14 (Gaps) If the bodies of two cells are not adjacent, but are at a distance $< B$ then the space between them is called a *small gap*. We also call a small gap such a space between the

bodies of two colonies. On the other hand, if the distance of the bodies of two colonies is $> B$ but $< QB$ then the space between them is called a *large gap*. ⌟

Before the transfer phase, the base colony-pair consists of two colonies, having cells of kind Member_0 and Member_1. There may be a bridge between them, of $< Q$ cells of kind Bridge adjacent to each other, and extending one of the base colonies (see Section 4.3).

The transfer phase moves the base colony-pair left if $Drift = -1$, and right, if $Drift = 1$ in the left base colony. Consider $Drift = -1$ first. The kind of cells of the left base colony will turn from Member_0 to Member_1. Then a bridge is built to the left (extending the -1 value on the *Drift* track). This bridge can override an opposite old bridge ("old" meaning that its *Sweep* is maximal) or move into an empty area, kill other bridge cells or stem cells if they are not adjacent, while it extends. This bridge extension can finish in two ways:

- It stops at the boundary of another colony on the left (which consists of cells of type Member_0). (This can happen in the very first step, in which case no bridge is built.) Then this colony becomes the new left base colony, and the head moves to its left end (extending again the -1 value on the *Drift* track).
- The bridge reaches the size of Q cells. In this case, the bridge will be converted into the new left base colony, the kind of its cells becoming Member_0.

In the last sweep, in the old base colony-pair, if the majority of $Hold[j].Doomed$, $j = 1, \ldots, 3$, is 1 then turn the scanned cell into a stem cell: this is the way to destroy the colony representing a cell of M^* that must be killed..

The case $Drift = 1$ is analogous.

4.8 Booting

Ideally, the work of machine M starts from a single active cell-pair of the Booting kind, with addresses $Q - 1$ and 0, the middle cell-pair of a yet to be built colony-pair. The *Rider* track of the cell-pair holds a tape segment of the simulated Turing machine G, along with the simulated head. Such a cell-pair will be called a *booting pair*. The segment consisting of this cell-pair will be extended left-right by booting cells, eventually creating a colony-pair, as follows.

Main work Machine M performs Q times the following:

> Simulate G in the active pair on the *Rider* track for as many steps as the size of the represented tape segment of G, possibly stopping earlier if the represented head would exit the represented segment earlier. Move the active pair of M left or right as needed, adding cells to the segment, and giving them the appropriate addresses of their colonies. All new cells encountered must be stem cells (blanks), and all the ones in the segment already created must be of the Booting kind; otherwise call alarm.
>
> In all this, use zigging and feathering. Zigging uses small turns just as during simulation, but whenever a turn of G is simulated, *make a big turn* (as defined in Section 4.4.2), with the

necessary digressions (see Definition 4.9). When the active pair does not need moving, treat this as (say) a left turn from the point of view of feathering, making the necessary marking and digression. The simulation is continued only after the head returns to the active cell-pair from the digression.

Output If the simulated computation of G terminates (the *Output* field of simulated cell 0 of G is written) then write the same value to the *Output* field of cell 0.

Lifting If the Q steps are completed without the termination of the G computation, then create the colony-pair around the original pair of booting cells (and turn its cells into member cells). *Lift* (copy) the *Rider* track of its cells into the *Rider* track of the cell-pair simulated by it. Set the mode of the simulated cell-pair to Booting.

The only error-control during all this is the step counting (the cells are big enough to carry a counter up to Q), zigging, and the call of alarm mentioned in the main work above. This serves to notice when booting cells were introduced by a fault into a non-booting situation. We introduce no mechanism to correct faults during the booting phase, since we do not expect faults during it—see the probability analysis in Section 9.1.

5 Annotation

The definitions introduced here, characterizing trajectories in terms of their structural integrity, are needed both for defining the error-correcting part of the program and for its later analysis.

5.1 Health

Information on the *Info* track is protected by an error-correcting code. However, faults can ruin the simulation structure and disrupt the simulation (and the correctness of the error-correcting process) itself. Structural is maintained with the help of a small number of fields. The required relations among them allow the identification and correction of local damage. A configuration with local structural integrity will be called *healthy*.

Definition 5.1 (Homogenous domain) The tuple of fields

$$Core = (Kind, Drift, Addr, Sweep)$$

is called the *core*. An interval of non-stem adjacent neighbor cells is a *homogenous domain* if its core variables with the exception of *Addr*e the same value, *Addr* increases left to right. The *left end* of a domain is the left edge of its first cell, and its *right end* is the right edge of its last cell. ⌟

Remark 5.2 The track *ZigAddr* is also used in healing, but only in a limited way: when the zigging domain is extended over stem cells, then this track directs the head back to the front. It is not needed for the definition of homogenous domains and boundaries. ⌟

Definition 5.3 An *extended colony* is a colony with possibly an adjacent bridge extending it towards a neighbor colony. ⌟

Health is completely determined by the types of boundaries allowed between domains, so failures of health are locally detectable. Cells of a healthy configuration either form a domain of booting cells fitting into a single colony-pair, or are grouped into gapless extended colonies, with certain transitional intervals inside and between them. Defining health formally can be done mechanically on the basis of the informal descriptions given here, but the detailed description would be tedious.

- There is a base colony-pair, with possibly a bridge between them going in the direction of its own drift. If the sweep inside this colony-pair shows that the `Compute` rule is being performed then the sweep boundaries allowed inside the colony-pair are the front, and the edges according to past turns at the end of the interior (as in Definition 4.4) of the base colony-pair. Outside each edge is a domain with a lower sweep number.
- During the first sweep of transfer, the base colony (of the pair) in the direction of the drift is possibly extended by a bridge towards the target colony. This bridge may be in the process of consuming an earlier bridge extended in the opposite direction. In the later part of the first sweep of transfer, the bridge already reaches the target colony. If the bridge extends to a full colony then this is converted to the corresponding kinds of member cells in the second sweep of the transfer. Domains ahead and behind the front show the changes done by the transfer phase.
- Non-base colonies are called *outer colonies*. If an outer colony is not adjacent to its neighbor colony belonging to or closer to the base, then it is extended by a bridge that covers this gap.

Though we did not spell out formally all details of health, let us define a strengthening of this property.

Definition 5.4 (Super-health) Let ξ be a healthy configuration on an interval I that is also safe for turns (as in Section 4.4.2). We say that ξ is *super-healthy* if it ends in colonies on both sides and in each colony, whenever the head is not in the last sweep, the *Info* track contains a valid codeword as defined in Section 4.1. A tape configuration is *super-healthy* on I if it can be extended (by indicating the head position) to a super-healthy configuration on it. ⌟

Lemma 5.5 *In a healthy configuration, over any interval of size $< QB$ there are at most three boundaries between domains.*

Proof. There can be a boundary around the head, and possibly two colony-ends in case of a big gap between two neighbor colonies. □

Let us define outer cells and boundary types in more detail.

Definition 5.6 (Outer cells, desert) Let us denote by $\mathrm{Last}(d)$ the number of the last sweep of the work period. If a cell is stem or has $\mathit{Drift} = \delta$, $\mathit{Sweep} = \mathrm{Last}(\delta)$ then it will be called a *right outer cell* if $\delta = -1$, a *left outer cell* if $\delta = 1$. ⌟

According to this definition, a stem cell is both a left and a right outer cell. Recall Definition 4.5 of the front.

Definition 5.7 (Boundaries) A boundary of a homogenous domain is called *rigid* if its address is the end address of a colony in the same direction. A *boundary pair* is a right boundary followed by a left boundary at distance $< B$. It is a *hole* if the distance is positive. It is *rigid* if at least one of its elements is. ⌟

The health of a configuration ξ of a generalized Turing machine M is defined over a certain interval A. It depends on $\xi_{\text{tape}}(A)$, further on whether ξ_{pos} is in A and if it is, where. But we will mention the interval A explicitly only where it is necessary. In particular, some of the structures described above may fall partly or fully outside A. A tape configuration is called *healthy* on an interval A when there is a head position (possibly outside A) that turns it into a healthy configuration. Health only depends on the *Core* track and the lack of heal or rebuild marks on the tape. In a healthy configuration every cell's *Core* field determines the direction in which the front is found, from the point of view of the cell. A violation of the health requirements can sometimes be noted immediately:

Definition 5.8 (Coordination) The current cell-pair is *coordinated* if it is possible for them to belong to a healthy configuration. ⌟

In normal mode, if lack of coordination is discovered then the healing procedure is called. The following lemmas show how local consistency checking will help achieve longer-range consistency.

Lemma 5.9 *In a healthy configuration, each Core value along with Z determines uniquely the Core value of the other cell of the current cell-pair, with the following exceptions.*

- *During the first transferring sweep, while creating a bridge, the front can be a stem cell or the first cell of an outer colony.*
- *Every jump backward from a target colony can land on the last cell of a bridge (whose address is not recorded in the cell from which the head is jumping) or the last cell of the base colony.*

Proof sketch. To compute the values in question, use *ZigAddr* to find the address at the front, and refer to the properties listed above. □

Lemma 5.10 (Health extension) *Let ξ be a* tape *configuration that is healthy on intervals A_1, A_2 where $A_1 \cap A_2$ contains a whole cell body of ξ. Then ξ is also healthy on $A_1 \cup A_2$.*

Proof. The statement follows easily from the definitions. □

In a healthy configuration, the possibilities of finding non-adjacent neighbor cells are limited.

Lemma 5.11 *An interval of size $< Q$ over which the configuration ξ is healthy contains at most two maximal sequences of adjacent non-stem neighbor cells.*

Proof. By definition a healthy configuration consists of full extended colonies, with possibly stem cells between them. An interval of size $< Q$ contains sequences of adjacent cells from at most two such extended colonies. □

Lemma 5.12 *In a healthy configuration, a cell's state shows whether it is an outer cell, and also its direction towards the front. The Core track of a homogenous domain can be reconstructed from any of its cells.*

Proof. Whether a cell is outer is computed from some fields. If the cell is outer then *Drift* shows its direction from the front, else the parity of *Sweep* shows it. For booting cells, the number of simulation steps performed increases towards the front. □

5.2 Annotation, scale-up

Let us first define the notion of "almost healthy" (admissible) for trajectories. Informally, an admissible configuration may differ from a healthy one in a small number of intervals we will call "islands". It may also differ from a super-healthy one in somewhat larger intervals we will call "stains". Even a healthy configuration may contain stains: places in which the *Info* track differs from a codeword. Stains pose no obstacle to the simulation, and if they are small and few then will be eliminated by it, via the error-correcting code.

Definition 5.13 (Local configuration, replacement) Let ξ be a tape configuration and $\zeta(I)$ a local tape configuration on interval I. Then the tape configuration $\xi|\zeta(I)$ is obtained by replacing ξ with ζ over the interval I. ⌟

Definition 5.14 (Annotation) Consider a region R of space-time where for each time t the set of space-time points belonging to R is an interval $J(t)$. For constants

$$c_{\text{island}}, c_{\text{stain}} \tag{5.1}$$

to be specified later, an *annotated trajectory* over R is a tuple

$$(\eta, \mathcal{I}(\cdot), \mathcal{S}(\cdot)),$$

where η is a trajectory over R, further for each time t

a1) $\mathcal{I}(t)$ is a set of intervals called *islands*, each of size at most $c_{\text{island}}\beta B$, where $\eta(\cdot, t)$ can be changed into the healthy configuration by changing it only in the islands. The disorder of $J(t)$ in $\eta(\cdot, t)$ is covered by intervals of size $\leq (\beta + 2)B$ one inside each island.

a2) $\mathcal{S}(t)$ is a set of intervals called *stains*, each of size $\leq c_{\text{stain}}B$. Each island is contained in a stain, and $\eta(\cdot, t)$ can be changed into a super-healthy configuration by changing it only in the stains.

a3) For any extended colony, at a time separated by $24E\beta T$ from any faults, the set of stains consists of some stains outside its interior (as in Definition 4.4) and separated from each other by at least $ZB/3$, and in addition one more stain anywhere.

A trajectory is *admissible* over R if it allows an annotation there. Its *base colony-pair* at any time t is that of a healthy configuration into which it can be changed by changing the islands (its position is uniquely determined). The head is *free* when it is not in any island, the observed cell-pair is in normal mode, and is coordinated.

A configuration ξ is admissible if the one-time-step trajectory consisting of just ξ is admissible. ⌟

When considering a single time t we can refer to $\mathcal{I}(t)$ as $\mathcal{I}$ and $\xi(t) = \eta(\cdot, t)$ as ξ, and we can talk about the annotation of ξ. A configuration may allow several possible annotations; however, the codewords recoverable from it do not depend on the choice of the annotation.

Formally, the proof of error-correction will happen by proving that annotation can be extended forward in time; however, we will retain an informal language whenever it is clear how to translate it to annotation. Let us argue (informally, without replacing any later, formal argument), that local correction does not have to deal with more than three islands in any area of size QB.

Example 5.15 (Three islands) Suppose that the head has arrived at some colony C from the left, goes through a work period and then passes to the right. In this case, if no new noise occurs then we expect that all islands found in C will be eliminated by the healing procedure. On the other hand, a new island I_1 can be deposited (in the last pass).

Consider the next time (possibly much later), when the head arrives (from the right). If it later continues to the left, then the situation is similar to the above. Island I_1 will be eliminated, but a new one may be deposited. But what if the head turns back right at the end of the work period? If I_1 is close to the left end of C, then (due to the feathering construction) the head may never reach it to eliminate it; moreover, it may add a new island I_2 on the right of I_1.

When the head returns a third time (possibly much later), from the right, feathering the level of the simulated machine will cause it to leave on the left. Islands I_1, I_2 will be eliminated but a new island I_3 may be created by a new burst of faults before, after or during the elimination. So the healing procedure must count with possibly three islands possibly in close vicinity to each other.. ⌟

Let us now define formally the codes φ_{*k}, Φ^*_k needed for the simulation of history $(\eta^{k+1}, Noise^{(k+1)})$ by history $(\eta^k, Noise^{(k)})$. Omitting the index k we will write φ_*, Φ^*. To compute the configuration encoding φ_* we proceed first as done in Section 3.5, using the code ψ_* there, and then initialize the kind, sweep, drift and address fields appropriately. The value $Noise^*$ is obtained by a residue operation as in Definition 2.3; it remains to define η^*. In the parts of the history that can be locally annotated, and which we will call *clean*. if no colony has its starting point at x at time t, set $\eta^*(x,t) = \text{Vac}$. Otherwise $\eta^*(x,t)$ will be decoded from the *Info* track of this colony, at the beginning of its work period containing time t. More precisely:

Definition 5.16 (Scale-up) Let $(\eta, Noise)$ be a history of M. We define $(\eta^*, Noise^*) = \Phi^*(\eta, Noise)$ as follows. Consider position x at time t, and $J = (t - T^*, t]$. If x is not contained in some interval I such that η has an annotation over $I \times J$ then $\eta^*(x,t) = \text{Bad}^*$. Assume now that it is contained,

and let $\chi(\cdot, u)$ be some admissible history satisfying η over $I \times J$. If x is not the start of some colony C in χ then let $\eta^*(x,t) = \text{Vac}$; assume now that it is. Then let $t' \in J$ be the starting time in χ of the work period of C containing t, and let $\eta^*(x,t)$ be the value decoded from $\eta(C,t')$. In more detail, as said at the end of Section 4.1, we apply the decoding ψ^* to the interior of C to obtain $\eta(x,t)$. ⌟

This definition decodes admissible intervals and trajectories for η into histories of η^*. (We don't know yet whether they are trajectories of η^*.)

6 Healing and rebuilding

Here we define the part of the simulation program that corrects configurations of machine M that are "almost" healthy.

6.1 Stitching

We will show that a configuration admissible over an interval of size $> QB/2$ can be locally corrected; moreover, in case the configuration is clean, this correction can be carried out by the machine M itself.

Definition 6.1 (Substantial domains) Let $\xi(A)$ be a tape configuration over an interval A. A homogenous domain of size at least $7c_{\text{island}}\beta B$ will be called *substantial*. The area between two neighboring maximal substantial domains or between an end of A and the closest substantial domain in A will be called *ambiguous*. It is *terminal* if it contains an end of A. Let

$$\Delta = 39c_{\text{island}}\beta,$$
$$E = 6\Delta.$$

⌟

Lemma 6.2 *In an admissible configuration, each half of a a substantial domain contains at least one cell outside the islands. If an interval of size $\leq QB$ of a tape configuration ξ differs from a healthy tape configuration χ in at most three islands, then the size of each ambiguous area is at most ΔB.*

Proof. The first statement is immediate from the definition of substantial domains. By Lemma 5.5, there are at most 3 boundary pairs in χ at a total size of $3B$. Also, there are at most 3 islands of size $\leq c_{\text{island}}\beta B$. There are at most 5 non-substantial domains of sizes $< 7c_{\text{island}}\beta B$ between these: this adds up to

$$< (3 + 3 \cdot c_{\text{island}}\beta + 5 \cdot 7 \cdot c_{\text{island}}\beta)B < 39 \cdot c_{\text{island}}\beta B = \Delta B.$$

□

The following lemma forms the basis of the healing algorithm.

Lemma 6.3 (Stitching) *In an admissible configuration, inside a clean interval, let U, W be two substantial domains separated by an ambiguous area V. It is possible to change the tape on U, V, W using only information in U, W in such a way that the tape configuration over $U \cup V \cup W$ becomes healthy. Moreover, it is possible for a Turing machine to do so gradually, extending, and changing the tape in U or W or enlarging U or W gradually at the expense of V.*

The machine in question can make its turns via "small turns" as defined in Section 4.4.

Proof. We distinguish several cases, based on the kind of domains involved. At any step, if we find that $U \cup V \cup W$ is healthy then we stop.

1. Assume that U, W both belong to the same extended colony: either an extended base colony, or an outer extended colony, or the target.
   In this case, by Lemma 5.12, the content of the *Core* track of V in the healthy configuration is completely determined by that of U, W, So the intervals U and W, which will be recognized as the substantial ones as opposed to V, can be gradually extended towards each other in any order, overriding V. If U and W belonged to different sides of the front, then the new front will be the new boundary between U and W.
2. Suppose now that U, W do not belong to the same extended colony, but they are on the same side of the front: without loss of generality, to the left of it.
   In this case, only U can be outer or a target: suppose that it is outer. Now W is either in a target or in the base colony. If it is in the base colony then the base colony cannot have an extension to the left, because given that the front is to the right, the same sweep that created the extension would have created already a target, separating U (which being outer is not in the target) from W too much. So whether U is in the base colony or not, W is close to its colony's left end, which is in V. It must be extended to this left end. Following this, U must be extended until its reaches W. The *Sweep* values can be propagated without change, there is no need to coordinate them between U and W.

   Suppose that U is in a target, then W belongs to the extended base colony. Then U must be extended until its end-cell, and then W must be extended to meet U. Since both U and W are in the interior, the *Sweep* values must be equal, so they can be extended without change.
3. Suppose that U and W belong to different extended colonies, with the front between them.
   The *Sweep* values can be extended without change, there is no need to coordinate them between U and W.

   Suppose that U contains no bridge cells: in this case extend it to the right until it hits a colony end-cell. If you overlap W, decrease W accordingly. Due to admissibility, only bridge cells of W can be overwritten in this way (colonies don't overlap in a healthy configuration). Similarly, if W contains no bridge cells then extend it to the left, until it hits a colony end-cell. Again, only bridge cells of U can be overwritten in this way.

Only one of U and W can be in an outer extended colony, suppose that U is. If W is a target then we already extended it to its limit. Now extend the bridge of U to meet it. If W is in an extended base colony then extend its bridge until either meets U or reaches full length. In the latter case, extend a bridge of U to meet W.

If U is in a target then it is already at its right end: we extend a bridge of W to meet it.

□

6.2 The healing procedure

The healing and rebuilding procedures look as if we assumed no noise or disorder. The rules described here, however (as will be proved later), will clean an area locally under the appropriate conditions, and will also work under appropriately moderate noise.

Structure repair has two procedures. The first one, called *healing*, performs only local repairs of the structure: for a given (locally) admissible configuration, it will attempt to compute a satisfying (locally) healthy configuration. If it *fails*—having encountered an inadmissible configuration—then the *rebuilding* procedure is called, which is designed to repair a larger interval. To protect from noise, any one call of the healing procedure will change only a small part of the tape, essentially one cell: so a noise burst during healing will have limited impact.

Recall the parameters Δ, E introduced in Definition 6.1. Suppose that `Heal` is called at some position z. Then it sets

$$\mathit{Mode} \leftarrow \text{Healing},\ \mathit{Heal.Sweep} \leftarrow 1,\ \mathit{Heal.Addr} \leftarrow 0$$

in the current cell-pair. In accordance with the property of minimal change by *Heal*, these fields will only be used and updated in the current cell-pair; their values are not read from new cells onto which the head moves (in the traditional Turing machine model they would belong to the internal state).

h1) We start by surveying an interval R of at least $2E$ cells around the starting point z. (It will go a little farther than E cells in each direction, in search of an allowed turning point, due to feathering. If one is not found in 2β, it restarts.) Whenever the head steps on a stem cell or creates a new cell, *Drift* is set to point to z (to make sure that the head does not get lost in a homogenous domain of stem cells).

h2) During healing, we accumulate an interval of cells marked for rebuilding called a *germ*. The marking is done on a track different from *Core* and therefore not influencing consistency. If the procedure finds that the germ reaches size $4\beta B > 3\beta B$, then it calls rebuilding. Also if healing already finds its center at the front of a stretch of at least $4\beta B$ of the frontier zone of rebuilding—see later—then it just passes control to rebuilding, which will continue its expansion. If it finds itself between two opposing such fronts then the rightward rebuilding gets precedence.

h3) At the end of every call of healing, as long as there are inconsistencies to correct, a cell is added to this interval (increasing its size even if healing previously deleted one of the cells). On the other hand, if no more inconsistency is found then the germ will be decreased by one cell (erasing one of the marks at its end). This way, if healing succeeds in fewer than 4β steps then it eventually erases the germ; otherwise it starts rebuilding.

h4) Suppose that the survey finds a pair of substantial domains separated by an ambiguous area, at least Δ removed from the complement of R. Choose the ambiguous area closest to z. Perform the first needed "stitching" operation (a single step) as defined in Lemma 6.3. If the ambiguous area still remains, go to its cell closest to z and restart healing. Starting from a clean admissible configuration, this operation creates a new admissible configuration that is one step closer to being healthy.

If the required stitching operation is not found or fails—as can happen for example when the ambiguous area's size decreases to $< B$ and turns out not to be a legitimate boundary—then healing fails (just adding to the germ).

h5) Suppose that the whole surveyed area is found healthy. Then if the head is found to be in the frontier zone (as defined in Section 4.4.1), then return to normal mode, else move it one step towards the front and restart healing.

The healing procedure runs in $O(\beta)$ steps, whereas rebuilding needs $O(Q^2)$ steps.

Lemma 6.4 (Clean healing) *Assume that the head moves in a noise-free and clean space-time rectangle $[a,b)\times[u,v)$, with $b-a>6EB$, $v-u>24E\beta T$, touching every cell of $[a,b)$ at least once, and never in rebuilding mode. Then before time v, the area $[a+6\beta B, b-6\beta B)$ becomes healthy.*

Proof. If healing was not called then after the head touched every cell of $[a,b)$ the health of the area is proved. In healing, the head will not move away from an ambiguous area before eliminating it, creating a healthy interval I containing the two substantial intervals that originally bordered it. If the head leaves this interval in normal mode and the zigging does not start new healing then the healthy interval covered by zigging can be added to I to form an even larger healthy interval, and this continues until new healing starts. One of the substantial intervals of the new healing will be inside I, and when it is completed, I will be extended further. The head may later return into the healthy interval I, but it will then just continue the simulation without affecting health while staying inside.

The margins of size $6\beta B$ upper-bound the areas that healing surveyed but did not correct since subsequent calls moved its the center away. The time $48E\beta T$ allows 4β pairs of passes over an area of size $6E$ cells. □

6.3 Rebuilding

Rebuilding starts by extending a germ (created by healing) to the right, in a zigging way, expecting rebuild-marked cells on the backward zig. This notices if rebuilding started from a burst of faults

smaller than a germ in a healthy area, and calls alarm. Just as with healing, we will not mention disorder (since the program does not see it)—but the analysis will take disorder into account.

The notion of front, of Definition 4.5 will be extended also to the rebuilding mode. Here is an outline, for rebuilding that started from a cell z in the middle of a germ. The goal is that

r1) we end up with a new decodable area extending at least one colony to the left and one colony to the right of z.

r2) the process does not destroy any healthy colony (possibly just created by a recent incarnation of rebuilding).

The rebuilding operation uses its own addresses, but in a special way, to avoid being confused by the occasional deletion or insertion of cells. It uses two address tracks: $Rebuild.Addr_{-1}$ counts from the left end of the rebuilding area towards the head, and $Rebuild.Addr_1$ from the right end.

Here are the stages. Recall the stitching operation from Section 6.1.

Mark Starting from the germ, extend a rebuilding area over $3Q$ cells to the left and $3Q$ cells to the right from the center z. Besides the address tracks, use also a track *Rebuild.Sweep*. Every step that changes the configuration must be accompanied by zigging, to check that the rebuilding is indeed going on.

Rebuilding cannot assume anything about the content of the area encountered: in particular, it may represent the traces of some other rebuilding. Since a single burst of faults of size β (which cannot be completely excluded) could pass control to this other rebuilding process, this possibility must be handled.

There is only one way that the rebuilding process can fail in a clean area: big turn-starvation, as defined in Section 4.4.2.

Survey and Create More details of this stage will be given below. It looks for existing colonies (possibly needing minor repair) in the rebuilding area, and possibly creates some. As a result, we will have one colony called C_{left} on the left of z, one called C_{right} on the right of z, and possibly some colonies between them. Make all newly created colonies represent stem cells. Direct all the other colonies with drifts and bridges towards C_{left}. (The creation of a bridge may result also in the creation of a new colony if the bridge becomes Q cells long.) The interval covering C_{left} and C_{right} will be called the *output interval* of rebuilding. The pair of neighbor colonies with C_{left} on its left will be made the current colony-pair.

Mop Remove the rebuild marks, shrinking the rebuilding area onto the right end of C_{left}.

Marked cells from some interrupted rebuilding may remain even after the mop-up operation. These may trigger new healing-rebuilding later.

Remarks 6.5 1. Giving precedence to the right rebuilding has the drawback that one can design a initial configuration in which even in the absence of noise, level 1 structure will never arise even locally. Namely, we can fill the line with short intervals $\ldots, J_{-1}, J_0, J_1, \ldots$ each of which is the start (say of size $3Z$) of a right-directed marking process. Then the head, after moving left

on J_0, will be captured by the process on J_{-1}, then later captured by the similar process on J_{-2}, and so on. But we will not need to consider such pathological configurations.

⌟

Details of the Survey and Create stage

The complexity of this stage is due mainly to guaranteeing property (r2) above.

s1) Search in the marked area on left of z for a colony C, or a set of cells that looks stitchable into a colony by up to 3 healings. The first substantial domain should be at least 3β cells to the left of z, to make sure that C is *manifestly* to the left of z. If the search and the stitching attempt are successful, mark C as C_{left}. Try to perform the stitchings; if one fails then restart rebuilding.

Repeat this search for a colony manifestly to the right of z: if found, call it C_{right}.

s2) Suppose that only C_{left} is found, then create C_{right} on the right of z. The other case is symmetrical.

s3) Suppose that neither C_{left} nor C_{right} have been found. Then search the whole area, starting from the left, for a colony. If no such colony is found then create a pair of adjacent neighbor colonies $C_{\text{left}}, C_{\text{right}}$ around z.

s4) Suppose that both C_{left} and C_{right} have been determined (found or created). Then fill in the area between them. For this, first search from the left, for (stitchable) colonies, one-by-one. Once no more are found, fill in the areas between them with stem cells.

s5) Suppose that no colony has been found manifestly to the left or right of z, but one is found that is not such. Then either the left end is manifestly to the left or the right end is manifestly to the right of z. We will still work for satisfying goal (r1).

Suppose that the left end is manifestly to the left of z, then call the colony C_1. Create C_{left} on the left of C_1. Now search for another colony on its right. If not found, create C_{right}, manifestly on the right of z. If one is found that is not manifestly on the right then call it C_2 and create C_{right} on its right. The other case is symmetrical.

The steps above requiring the *creation* of colonies and bridges are destructive (the stitching ones are not). Therefore before a creation step, the whole survey preceding it is performed twice, describing the required action (erasing cells in order to build new ones in their place) by the same appropriate symbol in all surveyed cells, on two different tracks. These actions are then only performed if the two survey results are identical—otherwise the healing procedure is restarted from its center.

Lemma 6.6 *Suppose that a rebuilding procedure starts on one end of an admissible subinterval $J \subset I$ of a clean interval I, and runs with at most one burst of faults of size $\leq \beta$, to the finish (possibly restarted some times due to big turn starvation (see Section 4.4.2) without leaving I. Denoting by K the output interval of rebuilding, then $J \cup K$ will be admissible.*

Proof. If a burst occurs then no matter what mode it puts the machine into, zigging will discover the inconsistency and call alarm. Then according to part (h2) of the healing procedure the frontier zone of rebuilding will be discovered, and rebuilding will be allowed to continue.

The other event that may interrupt (restarting) rebuilding is big turn starvation. Assume for example that big turn starvation moves the head to the right repeatedly, but finally (since the head does not leave I) a big left turn is allowed to happen. Then over the area that has been passed during any of these rebuildings, there will be no big turn starvation preventing the head from turning right, since there the footprints of the big right turns (as define in Section 4.4.2) have now been spaced far from each other. So eventually a big enough area will be free from closely spaced footprints and a rebuilding succeeds. □

Now that our program is defined we can spell out a property announced in Section 4.4.2.

Lemma 6.7 *Suppose that a clean interval is passed over by the head without noise. Then it becomes safe for turns.*

Proof. The informal argument given in Section 4.4.2 can now be verified easily. □

Lemma 6.8 *Let P be a path with at most $Q/2E$ bursts of faults of size β over an interval $I = [a, b)$ that is admissible at the beginning of P. Then by the end of P, the subinterval* $\mathrm{Int}(I, 13QB)$ *will be still admissible. If no bursts occur on sections of the path that enter and exit I from the right then the subinterval $[a + 13QB, b)$ will also stay admissible.*

Proof. Let us divide the bursts of faults into three groups. As bursts intersecting I occur, let us call each island created by a new burst an *end-island* if it is closer than $2EB$ to either one of the ends of I or to an earlier end-island. Let us call the other islands within $7QB$ of the ends the *rebuilding* islands, and the rest the *interior* islands.

By definition the end-islands are confined to within QB of the ends of I. If the head enters into any colony away from the end-islands in normal mode, then it will correct any two islands that are within $2EB$ from each other (an isolated one that was there and a new one just created). A burst can create an island that is not corrected within one sweep of the simulation, only at the end of a big turn. These big turns are outside the interior of any extended colony and separated from each other by at least $ZB/3$, therefore the islands at their ends will be corrected in any later pass by the head that touches them.

Bursts can also affect rebuilding, but again the ones that do not become end-islands and are not corrected in one sweep are only the ones at the end of big turns. Even repeated rebuilding does not allow the remaining islands to be closer than $ZB/3$ to each other, therefore any new rebuilding process that touches one of these will correct it. It follows that if a rebuilding process is started at a distance of least $7QB$ from the edges then, since it does not reach any end-island, it will be able to complete, not reaching the interior $\mathrm{Int}(I, 13QB)$. Therefore the head will always reach this interior in normal mode.

The argument clearly proves also the last statement of the lemma, about a path with no faults on entering from the right. In this case there is at most one end-island on the right end. Indeed, if the path returns from the right end-colony, leaving an island there, then due to feathering by the simulated trajectory, next time it must move right from it, and by passing the island, must correct it. □

7 Isolated bursts

Here, we will prove that the healing procedure indeed deals with isolated bursts of faults. For the elimination of disorder created by faults we will rely on the Escape, Spill Bound and the Attack Cleaning properties of a trajectory in Definition 2.8.

Isolated bursts of faults don't create disorder larger than 3β. The head escapes a disorder interval I via the Escape property; while it is inside, the spreading of this interval is limited by the Spill Bound property. Every subsequent time when the head enters and exits I this gets decreased via the Attack Cleaning property, so it disappears after $O(\beta)$ such interactions—see Lemma 7.2 below.

In a clean configuration, whenever healing started with an alarm, the procedure will be brought to its conclusion as long as no new fault occurs. The trajectory properties, however, do not allow any conclusion about the state of the cell to which the head emerges from disorder. This complicates the reasoning, and may require several restarts of the healing procedure. By design, the healing procedure can change the *Core* track only in one cell. The following lemma limits even this kind of possible damage: it says that the head with the wrong information may increase some existing island, but will not create any new one.

Lemma 7.1 *In the absence of noise, no new island will arise.*

Proof. The islands are defined only by the *Core* track. In normal mode, this track changes only at the front. If this is not the real front, then we are already in or next to an island.

The healing procedure's change of *Core* is part of a stitching operation. Looking at the different cases of the proof of Lemma 6.3, we see that inside a healthy area, healing can only change the *Core* track in two ways. The first way is case 1, when the *Sweep* values were changed in a domain not belonging to the interior. Such an operation can be applied to any healthy configuration without affecting health. The second case is when the front is moved left or right. This does not affect health either. □

The following lemma is central to the analysis under the condition that bursts of faults are isolated.

Lemma 7.2 (Healing) *In the absence of noise in M^*, the history can be annotated. Also, the decoded history $(\eta^*, Noise^*)$ satisfies the Transition Function property of trajectories (Definition 2.8).*

Proof. In an annotated trajectory, call a space-time point a *distress event* if either a fault occurs there or the head steps onto an island. If after a distress event the head becomes free (according to Definition 5.14), then we will say that *relief* occurred. The extension of annotation is straightforward as long as no distress event is encountered. The Transition Function property is observed, as no obstacle arises to the simulation. Stains do not cause problems: the computation stage of the simulation cycle eliminates them using the error-correcting code.

We will bound the head in a space-time rectangle of size $O(\beta B \times \beta^3 T)$ before relief after any distress event. If a burst of faults occurs of size β then what we proved can be applied to the new situation. knowing that now we will be free faults for the desired time. This will just double the space and time bound for relief starting from the original distress.

With this bound, there are two possibilities: either the islands caused by the distress are eliminated, or the head moves away from them in the given time. The latter cannot happen in the middle of a sweep over some colony; indeed, suppose that the sweep is to the right. If the head leaves in the left direction from the islands then it will hit them immediately again as it tries to move the front right. If it leaves to the right then backward zigging will hit the islands immediately again. But the head can indeed leave the islands at the point where for example it has just turned left near the right end of some colony-pair. This can only happen once, due to the feathering property. Since they are not in the interior, these islands will not affect decoding and the computation of the transition function. At the end of the working period, the simulated head can turn left or right. If it turns right then any islands left at the right end will be eliminated during the transfer stage. It can turn left, but only once, due to the feathering property of the simulated machine.

Let us proceed to prove that relief happens within the given space and time bounds. At the time of the distress event, let us draw an interval I of size QB centered around the head. The head will not leave it before relief, so we will consider the changes of the configuration inside it. Let $S_1, \dots, S_m$ be the list of substantial domains of I, and I' the subinterval obtained by deleting S_1 and S_m. and $A_1, \dots, A_{m-1}$ the ambiguous areas between them. We have $m = O(1)$, and $\sum_i |A_i| = O(\beta B)$. In what follows the boundaries of these intervals may slowly change in time, and some of the A_i may get eliminated.

Every time the head is in the disorder it must leave within time $O(\beta^2 T)$, due to the Escape property. It can leave disorder only $O(\beta)$ times, since every time it leaves on the same edge as entering, the Attack Cleaning property decreases the disorder. When the head enters a clean area J, there are several possibilities.

- Rebuilding mode. Since every progress step of rebuilding is accompanied by zigging, the head either leaves J within one zig or alarm will be triggered within $O(\beta)$ steps.
- Healing mode. One call of healing either finishes in $O(\beta)$ steps or the head leaves J earlier. If a new call starts and finishes then J already contains I', the disorder has been eliminated, and we can refer to Lemma 6.4 to finish.

- Normal mode. If no alarm is triggered within 6β steps, then the head is in the frontier zone (as defined in Section 4.4.1), and is moving away from I': this implies relief. If alarm is triggered then a new healing starts.

Since these are the only possibilities, the bound $O(\beta B \times \beta^3 T)$ on the space-time rectangle until relief is proven. □

8 Cleaning

This section will scale up the Spill Bound, Escape, Attack Cleaning and Pass Cleaning properties of trajectories, proving them for the history $(\eta^*, Noise^*)$ decoded from a trajectory $(\eta, Noise)$.

8.1 Escape

Here will scale up the Escape property. Consider some fault-free path during a time interval J (later we will allow a single burst of faults of size β) over some space interval G of size $|G| = \lambda QB$. Intervals of time of length T we may consider as *steps*, since in a clean interval, the machine M will perform at least one step of computation during it.

Definition 8.1 For the times $t \in J$, let $K(t)$ denote set of all clean points in G. Let $K^*(t)$ be the union of subintervals of $K(t)$ that are super-healthy (see Definition 5.4) with

$$k(t) = |K^*(t)|.$$

The set $K(t)$ consists of maximal disjoint intervals. In the following lemmas, we consider one of these, $I(t)$ during a fault-free path, where it may slightly decrease (without disappearing), increase or merge with others. At the beginning of the consideration, has size nB with $n = \lambda Q$, $\lambda \leq 2\beta$. If the head leaves $I(t)$ within $4Zn$ steps then we will call its stay *short*, otherwise we call the stay *long*. ⌟

In a super-healthy clean interval in history η the simulation of η^* will proceed. In what follows we will estimate the growth of $k(t)$, due to the creation of colonies in $I(t)$ or adding new ones on an edge. Recall $U_k = U = Q^3$ in Definition 2.9. This is an upper bound on the number of computation steps in one work period, even allowing some calls for healing.

Lemma 8.2 *If $n < 2Q$ then the stay is short. Otherwise, every $2\lambda U$ steps of its stay increase $k(t)$ (the size of area of $I(t)$ "organized up to the next level") by at least QB.*

Proof. We will see that while the head does not leave $I(t)$, every certain number of steps results in some kind of progress.

1. Suppose that we are at some time when the rebuild procedure had started, from a base of rebuild-marked cells large enough not to result in alarm (renewed call for healing) after the first zigging. Then the rebuilding will be completed, increasing $k(t)$.

2. Suppose that we are at some time when the mode is normal. Then within $4Z$ steps of the simulation either alarm is called, or a step of progress will be made in the ordinary work of simulation. If the simulation performs a transfer operation and $n < 2Q$ then in moving new colony-pair, it must exit I.

3. Suppose that at some time alarm is called. Then according to the proof of Lemma 7.2, we either arrive at the above case 1 (failed healing) in $O(\beta^2 < Z)$ steps or healing finishes in normal mode with at least one step made either to move closer to the front or to make a move in the ordinary work of simulation.

   Indeed, if healing succeeds it leaves a healthy area. Lemmas 5.10 and 6.4 imply that the overlapping successful healing areas can be combined, so healing will not be repeated over the same area, and thus can slow down progress only by a factor $O(\beta^2)$ (negligible compared to the Z times slowdown by zigging), and even this in at most one sweep of simulation over any area.

4. Suppose that $n \geq 2Q$ and the stay is long. Without rebuilding, the continued healings add some colonies to the healthy area; any such colony will be turned super-healthy by the first complete work-period of the simulation, taking at most U steps of computation. If the head is already in a super-healthy area then it will perform the simulation of M^*. The simulated machine M^* has the same program as M, so it will also make switchbacks of at least $E > 4\beta > 2\lambda$ steps (for healing or zigging). Therefore every $2\lambda U \geq 4U$ computation steps simulating $\geq 2\lambda$ steps of M^*, will pass to the end of the super-healthy interval in which it is working, and whose size is at most λQB, and extend it—unless interrupted by rebuilding.

5. If any rebuilding starts then it will either finish and extend $K^*(t)$ by a colony, or will be interrupted by a big turn starvation, as defined in Section 4.4.2. Suppose that big turn-starvation happens at an attempted left turn: then the marked rebuilding area is already of a size $\geq 2QB$. Therefore a next big turn starvation cannot be due to an attempted right turn on the left, since the marking process does not proceed so far to the left. Repeated big turn starvations could only happen to the right, and the head would leave on the right before the assumed λU computation steps. Since this does not happen, rebuilding succeeds.

□

Now consider cases where the head exits or enters $I(t)$.

Definition 8.3 We will say that the right end of $I(t)$ is *advancing* at time t if the rightmost colony of $K^*(t) \cap I(t)$ is in the start of the process of transfer to the right. Advancing is defined similarly for the left end. ⌟

Lemma 8.4 *Consider a sequence of long stays $J_1, J_2, \ldots, J_r$ in $I(t)$, possibly interrupted by any number of short ones.*

a) *In a long stay, $k(t)$ will not decrease. If it does not increase by at least QB then the end on which the head leaves becomes advancing.*

b) *If the head enters on an advancing end for a long stay then $k(t)$ increases by at least QB before it leaves—even after any number of short stays in between.*

c) *If the number of steps spent on long stays in $I(t)$ is at least $s\lambda U$ then they will increase $k(t)$ by at least $(s/8)QB$.*

Proof. To (a): At the entrance from disorder, the head may slightly damage a colony. But the subsequent checking would discover it and the stay is long enough for a rebuilding to succeed, and even to create a new colony, increasing $k(t)$ by at least QB. In the absence of rebuilding, a simulation must have continued; but then it can exit only via a started transfer operation, creating an advancing end.

To (b): Suppose that the head leaves on a advancing right end, and then re-enters after possibly a number of short stays. Let C be the right member of the colony-pair that that started the right transfer operation. The head may destroy C when entering, but even after repeated short stays, it cannot go past it without either completing a rebuilding operation (and thus increasing $k(t)$) or completing the transfer operation, and entering C from the right from a new colony created by the transfer. The long stay suffices for either the transfer or a rebuilding to succeed.

To (c): Let us pair the long stays: (J_1, J_2), (J_3, J_4),.... If r is odd and J_r has $a\lambda U$ steps for $a \geq 2$ then by Lemma 8.2 it advances $k(t)$ by $\geq \lfloor a/2 \rfloor QB \geq (a/4)QB$. Similarly if the longer of J_{2i-1}, J_{2i} has $a\lambda U$ steps, then the pair has at most $2a\lambda U$ steps, the pair advances $k(t)$ by $(2a/8)QB$. Otherwise it follows from (a)-(b) that the pair increases $k(t)$ by $\geq QB$. □

The following lemma is the scale-up of the Escape condition.

Lemma 8.5 (Escape) *Let $c = 9$, so $2c + 1 = 19 = c_{\text{esc}}$ as defined in (2.5). In the absence of Noise*, the head will leave any interval G of size nB with $n = \lambda Q$, $1 \leq \lambda \leq 3\beta$, within $(2c + 1)\lambda^2 U$ steps.*

Proof. Consider a time interval of length $(2c + 1)\lambda^2 UT$. Suppose that a burst of faults of size $\leq \beta$ happens during it: then we will consider the larger part J of before or after the burst (or the whole interval if there is no fault). Let

$$d = c_{\text{pass}}/3, \; g = \lceil c_{\text{esc}}(7d)^2 \rceil.$$

We will consider a sequence of times

$$t_i = t_0 + igT,$$

where t_0 is our starting time. The time intervals $(t_i, t_{i+1}]$, each comprising g steps, will be called *skips*. Consider adjacent space intervals $L_1, L_2, \ldots, L_7$ of size dB. If the head is in L_4 then by the Escape property of trajectories it will escape $L_1 \cup \cdots \cup L_7$ within time gT, and in the process pass either over $L_1 \cup L_2 \cup L_3$ or over $L_5 \cup L_6 \cup L_7$. In the long run, counting such passes will help applying the Pass Cleaning property. Let us partition the interval G into subintervals L_j of form $[a, b)$ of size dB called *blocks*; the last one can be smaller. From the above, it follows that each skip

passes over at least 3 blocks (starting before them and ending after them), either in the left or in the right direction.
We will show the contribution to the growth of $|K(t)|$ made by the various events as they occur during various kinds of skip. Clearly it cannot grow larger than $|G|$ while the head stays in G. Let us call a skip *clean* if in it the head does not touch disorder in any L_j, otherwise we will call it *disordered*.

1. The number of disordered skips is at most $18\lambda Q(\pi/d+1)$.
   Proof. If $L = L_{j+2}$ in a right pass for π values of j then the Pass Cleaning property implies that it becomes clean. Similarly for left sweep. So it can contain disorder only for 2π skips in which it is L_{j+2}.

   If $L = L_{j+1}$ for π values of j in left-right passes then the Pass Cleaning property implies that L_{j+2} becomes clean. After this, due to the Attack Cleaning property, if there is still disorder in L then each left-right pass decreases it by $B/2$. So L can contain disorder only for at most $2(\pi+d)$ skips with left-right passes, and the same number of skips for right-left passes, so at most $4(\pi+d)$ passes altogether.

   The same reasoning works for $L = L_{j+r}$ for any $r \neq 2$ when the head touches disorder in L. So the number of skips in which L is equal to any of the L_{j+r} for $r = 1,\dots,5$ and L contains disorder is at most $18(\pi+d)$. A bound on number of skips in which the head touches disorder in any L_j is obtained if we multiply this bound by the total number $\lambda Q/d$ of skips.
2. The number of clean skips happening during short stays (as in Definition 8.1) is at most $3\lambda^2 ZQ^2$.
   Proof. If a clean skip belongs to a stay in an interval $I(t)$ then this interval has length at least $c_{\text{pass}}B$. It follows that by the Attack Cleaning property, after exiting, the next entry either increases $|I(t)|$ by $\geq B/2$ or joins it to another interval in $K(t)$ of size $\geq B/2$. The total number of such joining events is at most $n = \lambda Q$. The number of increases by $B/2$ is also at most $2n$, so the number of such short stays is at most $4n$. In each short stay the number of steps is $\leq 4Zn$, so the number of skips is at most $(4/3)Zn$. Multiplying this by $4n$ gives the bound $(16/3)Zn^2 \leq 6\lambda^2 ZQ^2$.
3. The total number of steps spent in long stays is $> (c\lambda^2 - 1)U$.
   Proof. It follows from part 1 above that the number of steps in disordered skips is $\leq 18g\lambda Q(\pi/d+1)$. The number of steps in clean skips during short stays is $\leq 3g\lambda^2 ZQ^2$. So the number of steps of both kinds is still $< Q^3 = U$. Subtracting this from the total number $c\lambda^2 U$ of steps gives the estimate.

Lemma 8.4 implies from part 3 that the long stays increase $|K^*(t)|$ by $> (c\lambda/8 - 1/8\lambda)QB$. If $c = 9$ then this becomes $> |G| = \lambda QB$. □

8.2 Weak attack cleaning

This section will scale up the Attack Cleaning property of trajectories (Definition 2.8) to machine M^*, but first only in a weaker version, restricting the number of bursts of faults in the relevant

interval.

Definition 8.6 (Trap) Recall the definition of trains in Section 4.4.2. In an interval I, clean except possibly up to 3π islands of size β, a point is considered a *leftward trap* if (after changing it in the islands), it is at a right-directed frontier zone with a train having $FCount_1 \geq F$ (indicating an imminent left turn). ⌟

The Attack Cleaning property says the following for the present case. Let P be a path that is free of any fault of η^*. For current colony-pair (x, x') (where $x' < x + 2QB$), suppose that the interval $I = [x - c_{\text{marg}}QB, x' + QB)$ is clean for M^*. Suppose further that the transition function, applied to $\eta^*(x, t)$, directs the head right. Then by the time the head comes back to $x - c_{\text{marg}}QB$, the right end of the interval clean in M^* containing x advances to the right by at least $QB/2$.

Lemma 8.7 (Weak attack cleaning) *In addition of the above condition of attack cleaning, assume that the faults of size of trajectory η in the interval I covered by at most $s < Q/3E$ bursts of size β. Then the conclusion holds.*

Proof. The computation phase of the simulation on the colony-pair (x, x') is completed, then the transfer phase begins, possibly entering the disorder to the right of $x' + QB$. We argue that there are only two ways for the head to get back to $x - c_{\text{marg}}QB$.

at1) The transfer into a new colony-pair with starting point $y \geq x' + QB$ succeeds despite the disorder, and the clean interval extends over it, before the head moves left to $x - c_{\text{marg}}QB$ in the course of the regular simulation. Some inconsistencies may be discovered along the way, but they are corrected by healing.

at2) The inconsistencies encountered along the way trigger some rebuilding processes. Eventually a complete, clean rebuilding area is created, the rebuilding succeeds, leaving a clean colony also to the right of $x' + QB/2$.

By the Spill Bound property, disorder can spill left of $x' + QB$ only by $c_{\text{spill}}B$ as long as the path P is fault-free. If a burst of faults creates an island at a distance $\geq EB$ from the disorder and other islands then this island will be healed unless it is at a rightward trap. The feathering property assures that such remaining islands are at a distance $\geq ZB/3$ from each other, ready to be healed at any next pass. There are at most s bursts of size $\leq \beta$, so the possibly not correctable islands are all within distance sEB of the disorder, to the right of

$$R = x' + QB - (sE + c_{\text{spill}})B.$$

Consider this as the new left end of the disorder; the bound s on the number of bursts implies that size of the new spill is still $< QB/3$.

Suppose that rebuilding is not initiated (with creating a substantial germ): then the head can move deeper left into the colony of x' only by the normal course of simulation: the transfer stage of the simulation must be carried out, and this requires at least as many attacks to the right as the number of sweeps in the transfer stage. Every attack (followed by return) extends the clean

interval further, until the whole target colony becomes clean, and the transfer completed. This is the case (at1).

Recall the rebuilding procedure in Section 6.3: the rebuilding area extends $3Q$ cells to the left and right from its initiating cell. This may become as large as $6QB$ to the left and right. If initiated, its starting position z is to the right of R as defined above. It then may extend to the left to at most $z - 6QB$ (this is over-counting, since the cells of the colony of x are all adjacent): if the head moves to the left of this, then the rebuilding must have succeeded. Its many sweeps will result in attacks that clean an area to the right of the starting point. The procedure may be restarted several times, but those re-startings will also be initiated to the right of z. The rebuilding also must find or create a colony manifestly to the right of the restarting site. Since $R > x' + QB/2$ this will move the boundary of the area clean in M^* by at least $QB/2$: this is the case of (at2).

In the process described above, it is possible that the rebuilding finds a competing colony C starting at some $y \in x' + QB + [-\beta B, 0)$ which slightly (by the size of an island) overlaps from the right with the colony of x. The rebuilding may decide to keep C and to overwrite the rest of the colony of x' as a bridge (or even target). This does not affect the result. □

The following lemma draws the consequences of several applications of weak attack cleaning.

Lemma 8.8 *Let I_0 be an interval of size $> 28QB$, and J an adjacent interval of size kQB on its right. Assume that I_0 is super-healthy at the beginning of a path P whose faults are coverable by fewer than $Q/2E$ bursts over the interval $I_0 \cup J$, that passes I_0 at least 2^{k+14} times from left to right. Assume also that no faults occur on segments of the path that enter and exit I_0 from the right. Then at some time during the path, the interval J becomes admissible. The same statement holds if we switch left and right.*

Proof. We will apply Lemma 6.8 to intervals into which the I_0 and the right extensions of its admissible area. It is easy to see that $\mathrm{Int}(I, 13QB)$ will always stay admissible; at any time t let $I(t)$ be the largest admissible interval containing this. At the first pass, the rightmost colony of $I(t)$ will be at a distance $\leq 13QB$ from the right end. The lemma implies that the right end of $I(t)$ will not move left. Every time when P passes to the right of $I(t)$, there will be an attack, extending $I(t)$ by QB. However, not every time that path P passes I_0 to the right, will it necessarily pass also $I(t)$; it may turn back before. Feathering makes sure, however, that in 2^i passes, it moves right at least i times, so the 2^{k+14} passes are sufficient to include extend $I(t)$ over the whole interval J. □

8.3 Pass cleaning

We will assume that π is an even number. The scaled-up version of the Pass Cleaning property considers a path P with no faults of η^*, as it makes

$$\pi^* = \pi + 8 \tag{8.1}$$

passes over an the interval I of size $c_{\mathrm{pass}}QB$, and claims that they make $\mathrm{Int}(I, c_{\mathrm{marg}}QB)$ clean for η^*. The weak version of this property uses the additional assumption that the faults of P are coverable

by at most 3π bursts of size β. This is what we will prove first, so let us use this assumption. Since there are few bursts, there will be long intervals that are fault-free. Specifically, I is made up of clean subintervals of size $\geq 4ZB$ that we will call *basic holes* separated from each other and the ends by distances $\leq 12\pi ZB$.

The pass cleaning property of η cleans the basic holes (except for margins of size $\leq c_{\mathrm{marg}}B$). By the Spill Bound property, they may erode on the edges by a further amount $c_{\mathrm{spill}}B$. We will call these somewhat smaller intervals still basic holes. One more pass will make the basic holes, according to Lemma 6.7, safe for turns. The following two lemmas will show how some order will be established on them in two more passes. Recall that the maximum number of cells in a healing area, $E = O(\beta)$ from Definition 6.1, is a constant, much smaller than the zigging distance (in cell widths) defined in (4.2) as $Z = \pi^{1+\rho}$.

Definition 8.9 A clean interval J of size $> 3ZB$ not containing the head is called *right-directed* if it is safe for turns, the head is to the right of J, and its cells, maybe with the exception areas of size $3\pi EB$ on the ends, point towards a front at the right end of J (in normal operation or rebuilding), with the corresponding frontier zone inside J. ⌟

Lemma 8.10 *Consider an interval $[a, b)$ of size $\geq 4ZB$ that is safe for turns. If a path passes it noiselessly from left to right then it will leave a clean right-directed interval $[a, b')$ with $b' \geq b - c_{\mathrm{spill}}B$. The same is true when interchanging left and right.*

Proof. Assume that the starting mode is normal. If it remains normal then J naturally becomes directed by the time the head exits. Since it exits at the front (more precisely at the right of the frontier zone), the frontier zone stays behind. Let us see that also in all other cases when the head exits it leaves behind the right frontier zone.

If healing gets triggered then as long as healing succeeds it extends a healthy area. It moves towards right only if the front is on the right. If it does not succeed then rebuilding gets triggered, and since it does not touch the left end anymore, it either succeeds or exits on the right while trying to extend towards the right. The same considerations work if the starting mode is healing, except possibly on the part of size $\leq EB$ of the left end where the first healing took place, whose purview was not completely contained in J. □

Recall the definition of the feathering parameter F in (4.2).

Lemma 8.11 *Consider a fault-free path.*

a) *Suppose that an interval J is right-directed, and the head enters it from the right. If the head leaves on the right then J will stay right-directed.*

b) *If the head leaves on the left then within $2Z$ cells of the right end of J there will be a footprint of a big left turn in J (as defined in Section 4.4.2).*

The same conclusions hold if we switch left to right.

Proof. To (a): Both in normal operation and rebuilding, the head moves the front with itself and returns to it from every zig, except when it is captured at an end of J (by faults or disorder). If an island is encountered or faults occur then this will be healed, except when the healing interval (whose maximum size is EB) intersects the boundary of J (where the disorder may capture the head).

To (b): the front can turn back only at leftward traps; otherwise the simulation or rebuilding will move it forward, and the forward zigging prevents faults or disorder from turning it back prematurely. On turning back, it will leave behind a footprint of a big left turn, though the at most 3π bursts of size $\leq \beta$ covering the faults happening in between can shorten this train (of size $Z/2$) by $3\pi \ll Z$ cells.

□

Lemma 8.12 (Weak pass cleaning) *Suppose that a path P has no faults of η^*, it makes π^* passes over the interval I starting from the left, with its faults covered by at most*

$$s = \pi^*(\pi^* + 2^{c_{\text{pass}}+c_{\text{marg}}+14}).$$

bursts of size β in I. Then by end of the π^*th pass the interior $\text{Int}(I, c_{\text{marg}}QB)$ *becomes clean for η^*.*

Proof. Let us number the passes after the first π of them like pass 1,2,3,....

1. The first $\pi + 1$ passes make the basic holes clean and safe for turns, except for margins of size $\leq (c_{\text{marg}} + c_{\text{spill}})B$.

   Proof. As shown above, the basic holes, of size $\geq 4ZB$ and separated from each other and the ends by distances $\leq 4sZB$, become and stay clean for η in the first π passes, except for margins of size $\leq (c_{\text{marg}} + c_{\text{spill}})B$. Now the next pass, called pass 1, will make the basic holes safe for turns.

Assume that pass 1 was from right to left; otherwise, we start one pass later. Since we will finish in 7 passes we will still finish by $\pi^* = \pi + 8$. Let us call at any time a subinterval of I a *hole* if it is clean and safe for turns with the possible exception of s islands, and is maximal with this property. Holes will grow from basic holes.

2. Passes 2 and 3 will leave each hole left-directed, with the footprint of a left turn on the left end of each but possibly the last one.

   Proof.

   - Lemma 8.10 shows that the next pass turns each basic hole into a right-directed hole, with possibly a single island caused by faults.
   - Lemma 8.11(a) shows that the limited number of bursts of size β between this and the next pass that cover the faults, keeps the holes right-directed (just possibly adding some islands).
   - Lemma 8.10 shows again that the following leftward pass turns each hole into a left-directed one. Lemma 8.11(b) shows that this same leftward pass leaves a footprint of a left turn within EB of the right end of each hole.

3. After pass 4 (which is from the left), each interval between holes has a footprint of a big left turn on its left and one of a big right turn on its right.
   Proof. Feathering allows the footprint of a big left turn on the right end of a hole J to be erased only if it is moved to a distance at least $\geq FB$ to he right. As $F \gg$ the upper bound sZ on the separation between the holes, the process erases the disorder between J and the next one, J' unless J' contains a rightward trap at its end. In the latter case, a footprint of the big left turn at the right end of J remains. Lemma 8.11(b) shows that a footprint of a big right turn is created on the left end of every remaining hole.
4. Passes 5 and 6 make the interval clean and safe for turns, except possibly one island of size β caused by faults during pass 6.
   Proof. Having each boundary between holes surrounded by the footprints of big left and right turns, the next pass (which is from the right) will erase all these boundaries. So it makes the whole interval (other than the edges) clean. The following pass will make it safe for turns.
5. Pass 7 will clean $\mathrm{Int}(I, 7QB)$ for η^*.
   Proof. The intrusions from right to left into I before the next pass may create up to s islands of disorder, of size β, but these islands are placed at least $ZB/3$ apart. Indeed, in order for an island not to be cleaned, it must be at the right end of a big left turn, and so will leave a left turning footprint. So a later intrusion can only leave an island if it does not see $2E$ cells of this footprint and is therefore at a distance $> ZB$ to its left, or passes it by a distance FB. These intrusions create no rightward trap.

   When the full rightward pass comes, the head cannot pass without either already having an area already clean for η^* or cleaning, healing and rebuilding. No rebuilding started farther than $6BQ$ from the boundary will exit I: it may only result in alarm if it encounters disorder or the single burst of faults of size $\leq \beta$ occurs. As seen above, any disorder encountered at a time will be of size $\leq \beta B$, at a distance at least $ZB/3$ from the next one. Hence once the healing restarts after this disorder is cleaned, by part (h2), it will discover the rebuilding front, allowing it to continue. So eventually the rebuilding succeeds.

   This process may be repeated as the head moves right, and indeed the head cannot pass through I without continuing this process, making it super-healthy. The only missed areas are those that may contain a rebuilding interval reaching the edge of I.

   □

Lemma 8.13 *Consider the statement of Lemma 8.8 with the interval I_0 having the same length kQB with $k = c_{\mathrm{pass}} + c_{\mathrm{marg}}$ as the interval J. The conclusion holds also without the bound on the number of bursts covering the faults over the interval $I_0 \cup J$.*

Proof. Let $J_1 = I_0$, $J_0 = J$. We will show that if the conclusion does not hold then the paths passes over all the infinite sequence of consecutive adjacent intervals $J_2, J_3, \ldots$ on the left of J_1, of size $|J_1|$. Since the path is finite, this leads to a contradiction. Let $i = 1$.

1. Suppose the conclusion of Lemma 8.8 does not hold. Then the faults needed at least $Q/2E$ bursts of size β to cover them over $J_{i+1} \cup J_i$ during this time, consequently at least $\pi^* + 2^{k+14}$ bursts happened during some two consecutive pair of the 2^{k+14} rightward passes over J_{i+1}.
2. By the Escape property, each fault is covered in a burst of size β contained in a segment covering an interval $> 3c_{\text{pass}}QB$ with no more faults outside the burst. Since these segments don't pass over J_{i+1}, each contains a fault-free pass over J_{i+2}.

   Suppose that $\text{Int}(J_{i+2}, c_{\text{marg}}QB)$ becomes clean at some time during the first π^* of these passes. There are still 2^{k+14} fault-free passes over J_{i+2}, so we are back at the situation of part 1 with $i \leftarrow i+1$.
3. Suppose that J_{i+2} does not become clean during the first π^* passes. Then by Lemma 8.12, since $k = c_{\text{pass}} + c_{\text{marg}}$, the number set of faults need a number bursts of size β covering them in J_{i+2} exceeding $s = \pi^*(\pi^* + 2^{k+14})$. Then at least $\pi^* + 2^{k+14}$ bursts happen over J_{i+2} between some consecutive pair of the left-right passes over J_{i+2}. By the Escape property, each burst is contained in a segment covering an interval $> 3c_{\text{pass}}QB$ with no more fault outside the bursts. Since these segments don't pass over J_{i+2}, each contains a fault-free pass over J_{i+3}. This brings us back to the situation of part 2 with $i \leftarrow i+1$.

□

Let us remove the bound on the number of bursts in the Pass Cleaning property of η^*

Lemma 8.14 (Pass cleaning) *Let P be a space-time path without faults of η^* that makes at least π^* passes over an interval I of size $c_{\text{pass}}QB$. Then there is a time during P when $\text{Int}(I, c_{\text{marg}}QB)$ becomes clean.*

Proof. Let $J_1 = I$. We will show that if the conclusion does not hold then the paths passes over all the infinite sequence of consecutive adjacent intervals $J_2, J_3, \ldots$ on the left of J_1, of size $|J_1|$. Since the path is finite, this leads to a contradiction. Let $i = 1$, $k = c_{\text{pass}} + c_{\text{marg}}$.

1. By weak pass cleaning (Lemma 8.12), if $\text{Int}(J_i, c_{\text{marg}}QB)$ did not become clean for η^*, the number of bursts of size β in J_i needed to cover the faults is more than $s = \pi^*(\pi^* + 2^{k+14})$. Therefore there is a time interval between two consecutive left-right passes over J_i with at least $2(\pi^* + 2^{k+14})$ bursts over J_i. Due to the Escape property, each part of P containing one of these bursts has a noise-free segment of size $> 2c_{\text{pass}}QB$ either on the left or on the right of J_i. Without loss of generality we can assume that at least half of them are on the left, giving $\pi^* + 2^{k+14}$ noise-free passes over J_{i+1} during this time.
2. If J_{i+1} does not become clean during the first π^* of these passes then restart the reasoning, going back to part 1, setting $i \leftarrow i+1$. Otherwise by Lemma 8.13, interval J_i becomes clean during the next 2^{k+14} noise-free passes over J_{i+1}, contrary to the assumption.

□

8.4 Attack cleaning and spill bound

Let us remove the bound on the number of bursts in the scale-up of the Attack Cleaning property.

Lemma 8.15 (Attack cleaning) *Consider the situation of Lemma 8.7. The conclusion holds also if the number of bursts of size $\leq \beta$ needed to cover the faults of η in $I = [x - c_{\text{marg}}QB, x' + QB)$ is not bounded by $Q/3E$.*

Proof. By Lemma 8.7, we need now only to consider the case when there are at least $Q/3E$ bursts. Consider the path $P' \subseteq P$ containing the first $\pi^* + 2^{c_{\text{marg}}+16}$ of these. The Escape property implies that P' passes the interval J of length $c_{\text{pass}}QB$ on the right of I this many times. The Pass Cleaning property then implies that $\text{Int}(J, c_{\text{marg}}QB)$ becomes clean for η^* during P'. Then Lemma 8.13 (applied in the left direction) implies that within the next $2^{c_{\text{marg}}+16}$ passes, the disorder of η^* of length $\leq (1 + c_{\text{marg}})QB$ between the old clean interval ending at $x' + QB$ and the new one beginning at $x' + (c_{\text{marg}} + 1)QB$ will be erased. □

Let us prove the scaled-up version of the spill bound property.

Lemma 8.16 (Spill bound) *Suppose that an interval I of size $> 2c_{\text{spill}}QB$ is clean for η^*. and let P be a path that has no faults of η^*. Then $\text{Int}(I, c_{\text{spill}}QB)$ stays clean for η^*.*

Proof. Without loss of generality, consider exits and entries of the path on the left of I. Let C_0, C_1 be the two leftmost colonies in I, where by definition C_0 is at the very end of I. We can assume that C_0 is damaged since the Spill Bound property of η allows a spill of size $c_{\text{spill}}B$ into it. Consider a part of the path entering I and then leaving again on the left. As long as disorder is at least at a distance EB away from C_1, when the path enters these colonies it just continues the simulation. Any burst of faults will be corrected or leave an island subject to the limitations of health.

The islands created by faults in C_0 can be divided into two groups: one is a group starting from the left end in such a way that the distance between consecutive elements is at most EB.

The other ones are at a distance $\geq ZB/3$ from each other, at left turning footprints. It follows that if the faults can be covered by at most $Q/3E$ bursts of size β in P over C_0 then no colony on the right of C_0 will be damaged, since the first group never passes beyond C_0. On the other hand, if than $Q/3E$ bursts are needed in C_0 then we can finish just is in the proof of Lemma 8.15. □

9 Proof of the theorem

Above, we constructed a sequence of generalized Turing machines $M_1, M_2 \ldots$ with cell sizes $B_1, B_2, \ldots$ where M_k simulates M_{k+1}. The sequences and dwell periods were also specified in Definition 2.9. Here, we will use this construction to prove Theorem 1.

9.1 Fault estimation

The theorem says that there is a Turing machine M_1 that can reliably (in the defined sense) simulate any other Turing machine G. Before the simulation starts, the input x of G must be encoded by a code depending on its length $|x|$. We will choose a code that represents the input x as the information content of a pair of cells of M_{k_0} for an appropriate $k_0 = k_0(x)$, and set their kind to Booting. Note that the code does not depend on the length of the computation to be performed. At any stage of the computation there will be a highest level K such that a generalized Turing machine M_K will be simulated, with its cells of the Booting kind.

As the computation continues and the probability of some fault occurring increases, the encoding level will be raised again and again, by lifting mechanism of Section 4.8. Let $\mathcal{E}_k$ be the event that no burst of level k occurs in the space-time region $[-2B_k, 2B_k) \times [0, T_{k+1})$. This implies that the history on level k can be annotated in this region. Let

$$\mathcal{D}_k = \bigcap_{i<k} \mathcal{E}_i \cap \neg\mathcal{E}_{k+1},$$

then $\mathsf{P}(\bigcap_k \mathcal{E}_k) \geq 1 - \sum_k \mathsf{P}(\mathcal{D}_k)$. Let us show

$$\sum_k \mathsf{P}(\mathcal{D}_k) = O(\varepsilon). \tag{9.1}$$

The number of top level steps on level k is at most $3Q_kF_k$, where F_k is the feathering digression size defined in (4.2). Indeed, in Section 4.8 machine M_k performs at most Q_k simulation steps, but the feathering with big turns may introduce digression steps of up to $2F_k$ per turn.

From Definition 2.9 feathering constant F_k of (4.2) is obtained as

$$F_k = (8k + c_2)^{3+2\rho} < k^4$$

if ρ is a small enough constant and k is large. Hence

$$3F_kQ_k \leq 3c_1^2k^4 \cdot 2^{1.2^k}.$$

Lemma 2.4 bounds the probability of burst of level k by $\varepsilon \cdot 2^{-1.5^{k-1}}$, giving

$$\mathsf{P}(\mathcal{D}_k) = O(\varepsilon k^4 2^{1.2^k - 1.5^{k-1}}),$$

which shows (9.1).

Suppose that machine G produces output at its step t (recall that there is no halting, but the output in cell 0 will not change further). Let t' be any time after the point in the work of machine M_1 at which the simulation of G reaches this stage. For each k let τ_k be the (random) last time before t' when the head of the simulated machine M_k reaches position 0. Let $\mathcal{E}'_k$ be the event that no burst of level k occurs in the space-time region $[-B_k, B_k) \times (\tau_k, \tau_k + T_k]$.

Then $\bigcap_k \mathcal{E}_k \cap \bigcap_k \mathcal{E}'_k$ implies that at time t' the *Output* field of cell 0 has the output of machine G. By an argument equivalent to the one given above, the probability of this event is $1 - O(\varepsilon)$.

Just as above, it is easy to see $\mathsf{P}(\cap_k \mathcal{E}'_k) \geq 1 - O(\varepsilon)$. (Even though the rectangles defining the events $\mathcal{E}'_k$ are random, a probability estimate can be obtained for $\neg\mathcal{E}'_k$ similarly to $\neg\mathcal{E}_k$ above: just average over all possible values of τ_k.)

9.2 Space- and time-redundancy

Even with the simple tripling error-correcting code, there is a constant $c > 1$ such that a colony of level k uses at most c times more space than the amount of information contained in the cell of level $k+1$ that it simulates. Therefore if k is the level that needs to be simulated before an output of G can be reached then the space used at that time is at most c^k times the space needed to just store the information. If G produces output at time t then this is about $c^k t$. The size of cells of level k is of the order of $2^{1.2^k}$, and this is also the the order of the number t of steps of G that can be simulated. So k is about $d \log\log t$ with $d \approx 1/\log 1.2$. This gives a space redundancy factor

$$c^k \approx c^{d \log\log t} = (\log t)^\alpha$$

for some $\alpha > 0$. The estimate for time redundancy is similar.

10 Discussion

A weaker but much simpler solution If our Turing machine could just simulate a 1-dimensional fault-tolerant cellular automaton, it would become fault-tolerant, though compared to a fault-free Turing machine computation of length t, the slow-down could be quadratic. (Such a solution would be only *relatively* simpler, being a reduction to a complex, existing one.) We did not find an easy reduction by just having the simulating Turing machine sweep larger and larger areas of the tape, due to the possibility of the head being trapped too long in some large disorder created by the group of faults. Trapping can be avoided, however, *provided that the length t of the computation is known in advance*. The cellular automaton C can have length t , and we could define a "kind of" Turing machine T with a *circular tape* of size t simulating C. The transition function of T would move the head to the right in every step (with any backward movement just due to faults).

Decreasing the space redundancy We don't know how to reduce the time redundancy significantly, but the space redundancy can be apparently reduced to a multiplicative constant. Following Example 4.3, it is possible to choose an error-correcting code with redundancy that is only a factor δ_k with $\prod_{k=1}^{\infty}(1-\delta_k) > 1/2$. This also requires a more elaborate organization of the computation phase described in Section 4.5 since the total width of all other tracks must be only some δ_k times the width of the *Info* track. For cellular automata, such a mechanism was described in [6].

Other models There is probably a number of models worth exploring with more parallelism than Turing machines, but less than cellular automata: for example having some kind of restriction on the number of active units. On the other hand, a one-tape Turing machine seems to be the simplest computation model for which a reasonable reliability question can be posed, in the framework of transient, non-conspiring faults of constant-bounded probability.

A simpler, universal computation model is the so-called *counter machine*. This has some constant number of nonnegative integer counters (at least two for universality), and an internal state. Each transition can change each counter by ± 1, depends on both the internal state and on the set of those counters with zero value. A fault can change the state and can change the value of any counter by ± 1. It does not seem possible to perform reliable computation on such a machine in any reasonable sense. The statement of such a result cannot be too simple-minded, since there is *some* nontrivial task that such a machine can do: with $2n$ counters, it can remember almost $2n$ bits of information with large probability forever Indeed, let us start the machine with n counters having the value 0, and the other n having some large value (depending on the fault probability ε). The machine will remember forever (with large probability) which set of counters was 0. It works as follows (in the absence of a fault): at any one time, if exactly n values have value 0, then increase each nonzero counter by 1. Otherwise decrease each nonzero counter by 1.

This sort of computation seems close to the limit of what counter machines can do reliably, but how to express and prove this?

Bibliography

[1] Eugene Asarin and Pieter Collins. Noisy turing machines. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, pages 1031–1042, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 1

[2] Charles H. Bennett. The thermodynamics of computation – a review. *Intern. J. of Theor. Physics*, 21:905–940, 1981. 1

[3] Ilir Çapuni and Peter Gács. A Turing machine resisting isolated bursts of faults. *Chicago Journal of Theoretical Computer Science*, 2013. See also in arXiv:1203.1335. Extended abstract appeared in SOFSEM 2012. 2.1

[4] Bruno Durand, Andrei E. Romashchenko, and Alexander Kh. Shen. Fixed-point tile sets and their applications. *Journal of Computer and System Sciences*, 78:731–764, 2012. 1

[5] Peter Gács. Reliable computation with cellular automata. *Journal of Computer System Science*, 32(1):15–78, February 1986. Conference version at STOC' 83. 1

[6] Peter Gács. Reliable cellular automata with self-organization. *Journal of Statistical Physics*, 103(1/2):45–267, April 2001. See also arXiv:math/0003117 [math.PR] and the proceedings of STOC '97. 1, 4.3, 10

[7] Peter Gács and John Reif. A simple three-dimensional real-time reliable cellular array. *Journal of Computer and System Sciences*, 36(2):125–147, April 1988. Short version in STOC '85. 1

[8] G. L. Kurdyumov. An example of a nonergodic homogenous one-dimensional random medium with positive transition probabilities. *Soviet Mathematics Doklady*, 19(1):211–214, 1978. 1

[9] Lulu Qian, David Soloveichik, and Erik Winfree. Efficient turing-universal computation with dna polymers. In Yasubumi Sakakibara and Yongli Mi, editors, *DNA Computing and Molecular Programming*, pages 123–140, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 1

[10] John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. Shannon and McCarthy, editors, *Automata Studies*. Princeton University Press, Princeton, NJ., 1956. 1